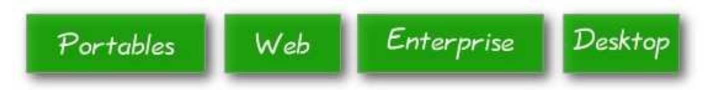
Introducción a wxPython

Una aplicación

Una aplicación es un programa de computadora que desempeña una tarea específica o un grupo de tareas. Navegador Web, reproductor de medios, procesador de textos son ejemplos de aplicaciones típicas. El término herramienta o utilidad es usado para una aplicación bastante pequeña y simple que desempeña una tarea simple. Un programa cp unix es un ejemplo de una herramienta. Todos estos forman juntos programas informáticos. El software informático es el más amplio término usado para describir al sistema operativo, datos, Programa de Computadoras, Aplicaciones, archivos mp3 o juegos de ordenador. Las aplicaciones pueden ser creadas para cuatro áreas diferentes.



Aplicaciones para compras en línea, wikis, weblogs son ejemplos de las APLICACIONES web más populares. Se accede allos con un Navegador Web. Ejemplos de Aplicaciones de escritorio incluyen Maya, Opera, Open Office o Winamp. La computación en la empresa es un área específica. Las aplicaciones en estas áreas son complejas y de gran tamaño. Las aplicaciones creadas para portátiles incluyen todos los programas desarrollados para teléfonos móviles, comunicadores, PDAs y otros similares.

Lenguajes de Programación

Actualmente hay varios Lenguajes de Programación muy utilizados. La siguiente lista se basa en el Índice de la Comunidad de programación TIOBE. Las cifras son de noviembre de 2010.

Position	lenguaje	Ratings
1	Java	18.5%
2	C	16.7%
3	C++	9.5%
4	PHP	7.8%
5	C#	5.7%
6	Python	5.7%
7	Visual Basic	5.5%
8	Objective C	3.2%
9	Perl	2.4%
10	Ruby	1.9%

Java es el lenguaje de programación más utilizado. Java se destaca en la creación de Aplicaciones portátiles móviles, programación de diversos aparatos y en la creación de Aplicaciones empresariales. Un cuarto de todas las aplicaciones está programado en C/C++. Son estándar para la creación de sistemas operativos y Aplicaciones de escritorio distintos. C/C++ son los Lenguajes de Programación de sistemas más utilizados. Las aplicaciones de escritorio más famosas fueron creadas en C++. Puede incluirse MS Office, Flash Macromedia, Adobe Photoshop o 3D Max. Estos dos idiomas también dominan el negocio de la programación de juegos.

PHP domina en la Web. Aunque Java es usado principalmente por las grandes organizaciones, PHP es usado por compañías más pequeñas e individuos. PHP es usado para crear Aplicaciones web dinámicas.

C # es el lenguaje de programación principal de la plataforma Microsoft. NET. C # es seguido en. NET por Visual Basic. Que representa un RAD popular. (Rapid Application Development o desarrollo rápido de aplicaciones).
Perl, Python y Ruby son los lenguajes de programación de “scripting” más utilizados. Comparten muchas similitudes y son competidores cercanos.
Objective C es el lenguaje de programación principal del ecosistema de Apple.

Python



Python es un lenguaje de "scripting" exitoso. Éste fue inicialmente desarrollado por **Guido van Rossum**. Éste fue por primera vez liberado en 1991. Python fue inspirado por los Lenguajes de Programación ABC y Haskell. Python es un lenguaje interpretado alto nivel, propósito general, multiplatforma. Algunos prefieren llamarlo un lenguaje dinámico. Éste es fácil de aprender. Python es un lenguaje minimalistico. Una de sus características más visibles es que no utiliza comas ni paréntesis. Python utiliza sangría en su lugar. En la actualidad Python es mantenido por un numeroso grupo de voluntarios en todo el mundo.

Para la creación de interfaces gráficas de usuario, los programadores de Python pueden elegir entre tres opciones decentes. PyGTK, wxPython y PyQt.

wxPython

wxPython es un kit de herramientas multiplataforma para la creación de Aplicaciones de escritorio GUI. El principal autor de wxPython es **Robin Dunn**. Con wxPython los desarrolladores pueden crear aplicaciones sobre Windows, Mac y sobre varios sistemas Unix. wxPython es una envoltura alrededor de wxWidgets, la cual es una biblioteca herramientas C++ multiplataforma madura. wxPython consiste de cinco módulos básicos.



El módulo **Controls** proveen los componentes comunes encontrados en aplicaciones gráficas. Por ejemplo un botón, una barra de Herramientas, o un Libro de Notas. Los componentes (Widgets) son llamadas controles bajo SO Windows. El módulo **Core** consiste de las clases elementales, que son usadas en desarrollo. Estas clases incluyen la clase Object, la cual es la madre de todas las clases, Dimensionadores (Sizers), los cuales son usados para diseño de componentes, Eventos, clases de geometría básica tales como Point (punto) y Rectangle (rectángulo). La Interfaz de Dispositivo Gráfico (**GDI**) es un conjunto de clases usado para dibujar sobre los componentes. Este módulo contiene clases para manipulation de Fuentes, Colores, Pinceles, Lápices o Images. El módulo **Misc** contiene otras clases variadas y módulo de funciones. Estas clases son usadas para registración (logging), configuración de aplicaciones, configuración de sistema, trabajar con mostrar (display) o joystick (palanca de control). El módulo **Windows** consiste de varias ventanas, que forman una Aplicación. Panel, Cuadro de Diálogo, Marcos o Ventanas Deslizantes.

wxPython API

API wxPython es un conjunto de métodos y objetos. Los componentes son esencialmente bloques de construcción de una GUI de la aplicación. Bajo Windows los componentes son llamados **Controls**. Podemos aproximadamente dividir los programadores dentro de dos grupos. Los que codifican aplicaciones y los que codifican bibliotecas. En nuestro caso, wxPython es una biblioteca que es usada por programadores de aplicaciones para codificar aplicaciones. Técnicamente, wxPython es un envoltorio sobre una API GUI en C++ llamada wxWidgets. Pero ésta no es una API nativa. p.e. no está escrita directamente en Python.

En wxPython tenemos gran cantidad de componentes. Estos pueden ser divididos dentro de algúnos grupos lógicos.

Componentes Básicos

Estos componentes proveen funcionalidad básica para componentes derivados. Éstos son llamados ancestros y no son por lo general directamente usados.



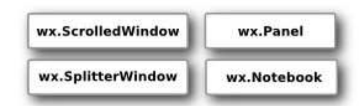
Componentes de nivel principal

Estos componentes existen independientemente de cualquier otro.



Contenedores

Los contenedores contienen otros componentes.



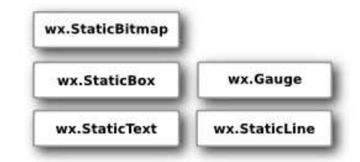
Componentes dinámicos

Estos componentes pueden ser editados por el usuarios.



Componentes Estáticos

Estos componentes visualizan información. Ellos no pueden ser editados por el usuario.



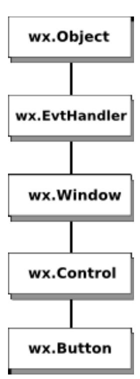
Otros componentes

Estos componentes implementan barra de estado, barra de herramientas y barra de menú en una Aplicación.



Herencia

Ésta es una relación específica entre componentes en wxPython. Esta relación es desarrollada por Herencia. La herencia es una parte crucial de la programación orientada a objetos. Los componentes forman una jerarquía. Los componentes pueden heredar funcionalidad desde otros componentes. Existen clases llamadas clases base, padres, o ancestros. Los componentes que heredan los llamamos componentes derivados, componentes hijos o descendientes. La terminología es prestada desde biología.



Herencia de un botón

Decimos que usamos un componente Botón en nuestras aplicaciones. El componente botón hereda desde 4 clases base diferentes. La clase más cercana es la clase wx.Control. Un componente botón es un tipo de ventana pequeña. Todos los componentes que aparecen en la pantalla son ventanas. Por lo tanto aquellos heredan desde *la clase wx.Window*. Estos son objetos invisibles. Ejemplos son dimensionadores, contexto de dispositivo o objeto local. Estas son también clases que son visible pero no son ventanas. Por ejemplo un objeto Color, objeto signo de intercalación o un objeto cursor. No todos los componentes son controles. Por ejemplo *wx.Dialog* no es un tipo de control. Los controles son componentes que son colocados sobre otros componentes llamada **Contenedores**. Es por eso que tenemos *una clase base wx.Control* separada.

Todas las ventanas pueden reaccionar a eventos. También lo hace el componente botón. haciendo clic en el botón, nosotros lanzamos el evento *wx.EVT_COMMAND_BUTTON_CLICKED*. El componente botón hereda al *wx.EvtHandler* mediante la *clase wx.Window*. Cada componente que reacciona a eventos debe heredar desde la clase *wx.EvtHandler*. Finalmente todos los objetos heredan desde la clase *wx.Object*. Esta es Eva, madre de todos los objetos en wxPython.

Esta fue una Introducción a wxPython.

Primeros Pasos

En esta parte del tutorial wxPython, vamos a crear algunos ejemplos simples.

Ejemplo simple

Comenzamos con un ejemplo muy simple. Nuestros primer guión (script) solo mostrará una ventana pequeña. No va hacer mucho. Vamos a analizar el guión línea por línea. Este es el código:

```
#!/usr/bin/python
# simple.py
import wx
app = wx.App()
frame = wx.Frame(None, -1, 'simple.py')
frame.Show()
app.MainLoop()
```

Éste es nuestro primer ejemplo en wxPython.

```
#!/usr/bin/python
# simple.py
```

La primer línea es una SHE-BANG (símbolo numeral seguido de signo de exclamación) seguido por la ruta a un interprete Python. La segunda línea es un comentario. Esto provee un nombre paral guión.

```
import wx
```

Esta línea importal módulo wxPython básico. Es decir, el núcleo, controles, gdi, misc y windows. Técnicamente wx es un namespace (espacio de nombres). Todas las funciones y objetos desde los módulos básicos se iniciarán con un prefijo ***wx.***. En la siguiente línea de código se creará un objeto Aplicación.

```
app = wx.App()
```

Cada programa wxPython debe tener un objeto aplicación.

```
frame = wx.Frame(None, -1, 'simple.py')
frame.Show()
```

Aquí creamos un objecto wx.Frame. El componente wx.Frame es un contenedor importante. Vamos a analizar este componente en detalle después. El componente wx.Frame es un componente padre para otros componentes. No tiene en si padres. Si especificamos None para un parámetro padre indicaremos que nuestro componente no tiene padres. Esto está al tope de la jerarquía de componentes. Después de que creamos El componente wx.Frame, deberemos llamar al método Show() que es el que en realidad lo muestra en la pantalla.

```
app.MainLoop()
```

En la última línea entra el mainloop. El bucle principal es es un ciclo sin fin. Esto captura y envia todos los eventos que existen durante la vida de nuestras aplicaciones.

Este fue un ejemplo muy simple. A pesar de esta simplicidad podemos hacer bastante con esta ventana. Podemos redimensionar la ventana, maximizarla, minimizarla. Esta funcionalidad requiere mucho de codificación. Todo esto esta oculto y provisto por omisión por el kit de herramientas de wxPython. No hay razón para reinventar la rueda.



Figura: simple.py

wx.Frame

El componente wx.Frame es uno de los mas importantes componentes en wxPython. Esto es un componente contenedor. Esto significa que puede contener otros componentes. En realidad puede contener cualquier ventana que no sea un Marco o un Cuadro de Diálogo. El wx.Frame consiste de una barra de título, bordes y un área central contenedora. La barra de título y bordes son opcionales. Aquellos pueden ser removidos por varias banderas (o flags o señales lógicas).

El wx.Frame tiene el siguiente constructor. Como podemos ver, este tiene siete parámetros. El primer parámetro no tiene un valor por omisión. Los otros seis parámetros lo tienen. Estos cuatro parámetros son opcionales. Los tres primeros son obligatorios.

```
 wx.Frame(wx.Window parent, int id=-1, string title='', wx.Point pos =
wx.DefaultPosition,
  wx.Size size = wx.DefaultSize, style = wx.DEFAULT_FRAME_STYLE, string name
= "frame")
```

wx.DEFAULT_FRAME_STYLE es un conjunto de banderas por omisión. wx.MINIMIZE_BOX | wx.MAXIMIZE_BOX | wx.RESIZE_BORDER | wx.SYSTEM_MENU | wx.CAPTION | wx.CLOSE_BOX | wx.CLIP_CHILDREN. Por combinación de varios estilos podemos cambiar el estilo del componente wx.Frame. A continuación un ejemplo corto.

```
#!/usr/bin/python
# nominimizebox.py

import wx
app = wx.App()

window = wx.Frame(None, style=wx.MAXIMIZE_BOX | wx.RESIZE_BORDER
        | wx.SYSTEM_MENU | wx.CAPTION |  wx.CLOSE_BOX)
window.Show(True)

app.MainLoop()
```

Nuestra intención fue mostrar una ventana sin una caja de minimizar. Por lo tanto no especificamos esta bandera en el parámetro estilo.

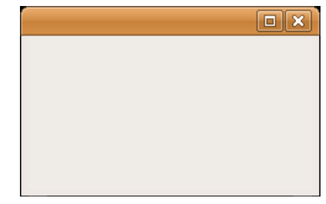


Figura: una ventana sin una caja de minimizar

Tamaño y Posición

Nosotros podemos especificar el tamaño de nuestras aplicaciones por dos caminos. Tenemos un parámetro tamaño (size) en el constructor de nuestro componente. O podemos llamar el método SetSize().

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# size.py

import wx

class Example(wx.Frame):
    def __init__(self, parent, title):
        super(Example, self).__init__(parent, title=title,
            size=(250, 200))
        self.Show()


if __name__ == '__main__':
    app = wx.App()
    Example(None, title='Size')
    app.MainLoop()
```

En este ejemplo, la aplicación podrá ser de tamaño 250x200 px.

```
def __init__(self, parent, title):
    super(Example, self).__init__(parent, title=title,
        size=(250, 200))
```

En el constructor fijamos el ancho del componente wx.Frame a 250px y el alto a 200px.

Del mismo modo, podemos posicionar nuestras aplicaciones en la pantalla. Por omisión la ventana es colocada en la esquina superior izquierda de la pantalla. Pero esto puede diferir sobre distintas plataformas de SO o incluso gestores de ventana . Algunos gestores de ventana colocan ventanas de aplicaciones por si mismas. Algunos de estos hacen alguna optimalización, de modo que las ventanas no se superpongan. Un programador puede posicionar la ventana por programación. Ya hemos visto un parámetro *pos* en el constructor de nuestro componente wx.Frame. Proveyendo otros distintos a los valores por omisión, podemos controlar la posición nosotros mismos.

Método	Descripción
Move(wx.Point point)	mueve una venta a la posición indicada
MoveXY(int x, int y)	mueve una venta a la posición indicada
SetPosition(wx.Point point)	fija la posición de una ventana
SetDimensions(wx.Point point, wx.Size size)	fija la posición y el tamaño de una ventana

Hay varios métodos para hacer esto.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# move.py

import wx

class Example(wx.Frame):

    def __init__(self, parent, title):
```

```
        super(Example, self).__init__(parent, title=title,
            size=(300, 200))

        self.Move((800, 250))
        self.Show()


if __name__ == '__main__':
    app = wx.App()
    Example(None, title='Move')
    app.MainLoop()
```

Esta es una situación particular. Puede ser que deseemos mostrar nuestras ventanas maximizadas. En este caso, la ventana es posicionada en (0, 0) y toma toda la pantalla. wxPython internamente calcula las coordenadas de la pantalla. Para maximizar nuestros wx.Frame, llamamos al método *Maximize()*.

Centrando en la pantalla

Si buscamos centrar nuestras aplicaciones en la pantalla, wxPython tiene un método práctico. El método Centre() simplemente centra la ventana en la pantalla. No se necesita calcular el ancho y el alto de la pantalla. Simplemente llamar al método.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# center.py

import wx

class Example(wx.Frame):

    def __init__(self, parent, title):
        super(Example, self).__init__(parent, title=title,
            size=(300, 200))

        self.Centre()
        self.Show()


if __name__ == '__main__':

    app = wx.App()
    Example(None, title='Center')
    app.MainLoop()
```

En este ejemplo, centramos una ventana pequeña sobre nuestra pantalla.

```
self.Centre()
```

Éste es el método que centra una ventana en la pantalla.

En este capítulo, tenemos creados algunos ejemplos simples de código en wxPython.

Menus y Barras de Herramientas

Una parte común en una GUI de la aplicación es una menubar. una barra de menú consiste de los objetos llamados menus. Los menús de nivel superior tienen sus etiquetas sobre la barra de menú. Los menús tienen items de menú. items de menú son comandos que desempeñan una acción específica dentro de las aplicaciones. Los menús pueden también tener submenúes, los cuales tienen sus propios items de menú. Las siguientes tres clases son usadas para crear barras de menú en wxPython. Las clases wx.MenuBar, wx.Menu y wx.MenuItem.

Menú simple

En nuestro primer ejemplo, vamos a crear una barra de menú con un menú archivo. El menu tendrá solamente un item de menú. Al seleccionar el item la aplicación sale.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

'''
ZetCode wxPython tutorial
Este ejemplo muestra un Menú simple.
author: Jan Bodnar
website: www.zetcode.com
last modified: September 2011
'''

import wx

class Example(wx.Frame):

    def __init__(self, *args, **kwargs):
        super(Example, self).__init__(*args, **kwargs)

        self.InitUI()

    def InitUI(self):

        menubar = wx.MenuBar()
        fileMenu = wx.Menu()
        fitem = fileMenu.Append(wx.ID_EXIT, 'Quit', 'Salir de la 
aplicación')
        menubar.Append(fileMenu, '&File')
        self.SetMenuBar(menubar)

        self.Bind(wx.EVT_MENU, self.OnQuit, fitem)

        self.SetSize((300, 200))
        self.SetTitle('Menú simple')
        self.Centre()
        self.Show(True)

    def OnQuit(self, e):
        self.Close()

def main():

    ex = wx.App()
```

```
    Example(None)
    ex.MainLoop()


if __name__ == '__main__':
    main()
```

Éste es un ejemplo pequeño con una mínima barra de menú funcional.

```
menubar = wx.MenuBar()
```

Primero creamos un objeto barra de menú.

```
file = wx.Menu()
```

A continuación creamos un objeto menú.

```
fitem = fileMenu.Append(wx.ID_EXIT, 'Quit', 'Salir de la  aplicación')
```

Agregamos un item de menú dentro del objeto menú. El primer parámetro es el id del item de menú. El id estándar se añade automáticamente un icono y un atajo de teclado. Ctrl+Q en nuestro caso. El segundo parámetro es el nombre del item de menú. El último parámetro define la cadena corta de ayuda que es mostrada sobre la barra de estado, cuando el item de menú es seleccionado. Aquí no hemos creado un wx.MenuItem explícitamente. Éste fue creado por el método Append() detrás de la escena. El método retorna el item de menú creado. Esta referencia podrá ser usada después para enlazar un evento.

```
self.Bind(wx.EVT_MENU, self.OnQuit, fitem)
```

Vinculamos el wx.EVT_MENU del item de menú al adaptado método OnQuit(). Este método cerrará las aplicaciones.

```
menubar.Append(fileMenu, '&File')
self.SetMenuBar(menubar)
```

Después de que, agregamos a menu dentro de la barra de menú. El caracter & crea un tecla aceleradora. El caracter que sigue al & es subrayado. De esta forma el menu es accesible via el atajo Alt + A. Al final, llamamos al método SetMenuBar(). Este método pertenece al componente wx.Frame. En el se fija la barra de menú.



Figura: Un simple ejemplo de Menú (en el código ejemplo se tradujeron tooltips al castellano)

Iconos y atajos

El ejemplo siguiente es esencialmente el mismo que el previo. En este momento, creamos manualmente un wx.MenuItem.

```
#!/usr/bin/python
```

```
# -*- coding: utf-8 -*-
'''
ZetCode wxPython tutorial
En este ejemplo, creamos manualmente un item de menú.
author: Jan Bodnar
website: www.zetcode.com
last modified: September 2011
'''

import wx

APP_EXIT = 1

class Example(wx.Frame):

    def __init__(self, *args, **kwargs):
        super(Example, self).__init__(*args, **kwargs)

        self.InitUI()

    def InitUI(self):

        menubar = wx.MenuBar()
        fileMenu = wx.Menu()
        qmi = wx.MenuItem(fileMenu, APP_EXIT, '&Quit\tCtrl+Q')
        qmi.SetBitmap(wx.Bitmap('exit.png'))
        fileMenu.AppendItem(qmi)

        self.Bind(wx.EVT_MENU, self.OnQuit, id=APP_EXIT)

        menubar.Append(fileMenu, '&File')
        self.SetMenuBar(menubar)

        self.SetSize((250, 200))
        self.SetTitle('Icons y shortcuts')
        self.Centre()
        self.Show(True)

    def OnQuit(self, e):
        self.Close()

def main():

    ex = wx.App()
    Example(None)
    ex.MainLoop()


if __name__ == '__main__':
    main()
```

En este ejemplo, creamos un item de menu de salida. Elejimos un icono personalizado y un atajo para el item de menú.

```
qmi = wx.MenuItem(fileMenu, APP_EXIT, '&Quit\tCtrl+Q')
qmi.SetBitmap(wx.Bitmap('exit.png'))
```

```
fileMenu.AppendItem(qmi)
```

Creamos a objeto wx.MenuItem. El caracter & especifica un acelerador de teclado. El caracter siguiente al ampersand es subrayado. El actual atajo es definido por la combinación de caracteres. Tenemos especificados los caracteres Ctrl+Q. Para que si apretamos Ctrl+Q, cerramos las aplicaciones. Ponemos un caracter de tabulado entre el caracter & y el atajo. De esta manera, logramos poner algún espacio entre ellos. Para proveer un icono para un item de menú, llamamos al método SetBitmap(). Un item de menú creado manualmente es añadido al menú al llamar el método AppendItem().

```
self.Bind(wx.EVT_MENU, self.OnQuit, id=APP_EXIT)
```

cuando seleccionamos el item de menú creado, el método OnQuit() es llamado.



Figura: Iconos y atajos

Submenúes y separadores

Cada menu pueden también tener un submenú. De esta manera podemos colocar comandos similares dentro de grupos. Por ejemplo podemos colocar comandos que ocultan/muestran barras de herramientas varias tales como barra personal, barra de direcciones, barra de estado o barra de navegación dentro de un submenú llamado *toolbars*. Dentro de un menú, podemos separar comandos con un separador que es una línea simple. Ésto es práctica común para separar comandos tales como New, Open, Save desde comandos tales como Print, Print preview con un separador simple. En nuestro ejemplo veremos, como podemos crear Submenúes y separadores de menú.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
'''
ZetCode wxPython tutorial
En este ejemplo, creamos un submenú y a separador de menu.
author: Jan Bodnar
website: www.zetcode.com
last modified: September 2011
'''

import wx

class Example(wx.Frame):

    def __init__(self, *args, **kwargs):
        super(Example, self).__init__(*args, **kwargs)

        self.InitUI()

    def InitUI(self):
```

```
        menubar = wx.MenuBar()

        fileMenu = wx.Menu()
        fileMenu.Append(wx.ID_NEW, '&New')
        fileMenu.Append(wx.ID_OPEN, '&Open')
        fileMenu.Append(wx.ID_SAVE, '&Save')
        fileMenu.AppendSeparator()

        imp = wx.Menu()
        imp.Append(wx.ID_ANY, 'Import newsfeed list...')
        imp.Append(wx.ID_ANY, 'Import bookmarks...')
        imp.Append(wx.ID_ANY, 'Import mail...')

        fileMenu.AppendMenu(wx.ID_ANY, 'I&mport', imp)

        qmi = wx.MenuItem(fileMenu, wx.ID_EXIT, '&Quit\tCtrl+W')
        fileMenu.AppendItem(qmi)

        self.Bind(wx.EVT_MENU, self.OnQuit, qmi)

        menubar.Append(fileMenu, '&File')
        self.SetMenuBar(menubar)

        self.SetSize((350, 250))
        self.SetTitle('Submenu')
        self.Centre()
        self.Show(True)

    def OnQuit(self, e):
        self.Close()

def main():

    ex = wx.App()
    Example(None)
    ex.MainLoop()


if __name__ == '__main__':
    main()
```

En el ejemplo de arriba, creamos los items de menú estándares New, Open y Save. Estos son separados desde un submenú con un separador horizontal. Un submenú tiene adicionalmente tres items de menú.

```
fileMenu.Append(wx.ID_NEW, '&New')
fileMenu.Append(wx.ID_OPEN, '&Open')
fileMenu.Append(wx.ID_SAVE, '&Save')
```

Aquí tenemos tres item de menú comunes. New, Open y Save.

```
fileMenu.AppendSeparator()
```

un separador de menú es añadido con el método AppendSeparator().

```
imp = wx.Menu()
```

```
imp.Append(wx.ID_ANY, 'Import newsfeed list...')
imp.Append(wx.ID_ANY, 'Import bookmarks...')
imp.Append(wx.ID_ANY, 'Import mail...')

fileMenu.AppendMenu(wx.ID_ANY, 'I&mport', imp)
```

Un submenú es también un wx.Menu. Tres items de menú son agregados al menú. El submenu es añadido al menú *file* con el método AppenMenu().

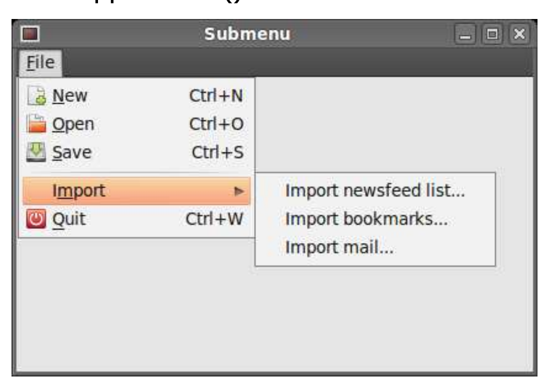


Figura: un submenú ejemplo

Item de menú de comprobación

Estos son tipos de árbol de items de menú.

- item normal
- item de comprobación
- radio item

En el siguiente ejemplo, demostraremos el item de menú de comprobación. Un item de comprobación de menú es visualmente representado por un tilde en el menu.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
'''
ZetCode wxPython tutorial
Este ejemplo crea un item de menú de comprobación.
author: Jan Bodnar
website: www.zetcode.com
last modified: September 2011
'''

import wx


class Example(wx.Frame):

    def __init__(self, *args, **kwargs):
        super(Example, self).__init__(*args, **kwargs)

        self.InitUI()

    def InitUI(self):

        menubar = wx.MenuBar()
        fileMenu = wx.Menu()
```

```
        viewMenu = wx.Menu()

        self.shst = viewMenu.Append(wx.ID_ANY, 'Show statubar',
            'Show Statusbar', tipo=wx.ITEM_CHECK)
        self.shtl = viewMenu.Append(wx.ID_ANY, 'Show toolbar',
            'Show Toolbar', tipo=wx.ITEM_CHECK)

        viewMenu.Check(self.shst.GetId(), True)
        viewMenu.Check(self.shtl.GetId(), True)

        self.Bind(wx.EVT_MENU, self.ToggleStatusBar, self.shst)
        self.Bind(wx.EVT_MENU, self.ToggleToolBar, self.shtl)

        menubar.Append(fileMenu, '&File')
        menubar.Append(viewMenu, '&View')
        self.SetMenuBar(menubar)

        self.toolbar = self.CreateToolBar()
        self.toolbar.AddLabelTool(1, '', wx.Bitmap('texit.png'))
        self.toolbar.Realize()

        self.statusbar = self.CreateStatusBar()
        self.statusbar.SetStatusText('Ready')

        self.SetSize((350, 250))
        self.SetTitle('Check item de menú')
        self.Centre()
        self.Show(True)


    def ToggleStatusBar(self, e):

        if self.shst.IsChecked():
            self.statusbar.Show()
        else:
            self.statusbar.Hide()

    def ToggleToolBar(self, e):

        if self.shtl.IsChecked():
            self.toolbar.Show()
        else:
            self.toolbar.Hide()

def main():

    ex = wx.App()
    Example(None)
    ex.MainLoop()


if __name__ == '__main__':
    main()
```

Tenemos una vista del menú, donde tenemos dos item de comprobación de menú. Estos dos items de

menú podrán mostrar y ocultar una barra de estado y una barra de Herramientas.

```
self.shst = viewMenu.Append(wx.ID_ANY, 'Show statubar',
    'Show Statusbar', tipo=wx.ITEM_CHECK)
self.shtl = viewMenu.Append(wx.ID_ANY, 'Show toolbar',
    'Show Toolbar', tipo=wx.ITEM_CHECK)
```

Si buscamos agregar un item de menú de comprobación, fijamos un tipo de parámetro para wx.ITEM_CHECK. El parámetro por omisión es wx.ITEM_NORMAL. El método Append() retorna un wx.MenuItem.

```
viewMenu.Check(self.shst.GetId(), True)
viewMenu.Check(self.shtl.GetId(), True)
```

Cuando las aplicaciones arrancan, tanto la barra de estado como la barra de herramientas son visibles. Lo que debemos comprobar tanto items de menú con el método Check().

```
def ToggleStatusBar(self, e):

    if self.shst.IsChecked():
        self.statusbar.Show()
    else:
        self.statusbar.Hide()
```

Mostraremos o ocultaremos la barra de estado de acuerdo al estado del Item de menú de comprobación. Nos encontramos con el estado del item de comprobación de menú con el método IsChecked(). Lo mismo con toolbar.

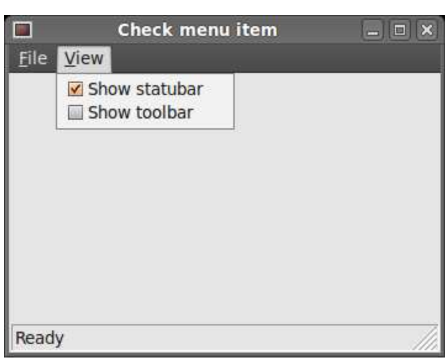


Figura: item de comprobación de menú

Menú contextual

Un menú contextual es una lista de comandos que aparecen bajo algunos contextos. Por ejemplo, en un Navegador Web Firefox, cuando hacemos clic derecho sobre una página web, tomamos un menú contextual. Aquí podemos recargar una página, volver o ver el código fuente de la página. Si hacemos clic derecho sobre una barra de Herramientas, tomamos otro Menú contextual para la gestión de barras de herramientas. Los menúes contextuales son algunas veces llamados menús emergentes.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
'''
ZetCode wxPython tutorial
En este ejemplo, creamos un menú contextual.
author: Jan Bodnar
website: www.zetcode.com
```

```
last modified: September 2011
'''

import wx

class MyPopupMenu(wx.Menu):

    def __init__(self, parent):
        super(MyPopupMenu, self).__init__()

        self.parent = parent

        mmi = wx.MenuItem(self, wx.NewId(), 'Minimize')
        self.AppendItem(mmi)
        self.Bind(wx.EVT_MENU, self.OnMinimize, mmi)

        cmi = wx.MenuItem(self, wx.NewId(), 'Close')
        self.AppendItem(cmi)
        self.Bind(wx.EVT_MENU, self.OnClose, cmi)


    def OnMinimize(self, e):
        self.parent.Iconize()

    def OnClose(self, e):
        self.parent.Close()


class Example(wx.Frame):

    def __init__(self, *args, **kwargs):
        super(Example, self).__init__(*args, **kwargs)

        self.InitUI()

    def InitUI(self):

        self.Bind(wx.EVT_RIGHT_DOWN, self.OnRightDown)

        self.SetSize((250, 200))
        self.SetTitle('Menú contextual')
        self.Centre()
        self.Show(True)

    def OnRightDown(self, e):
        self.PopupMenu(MyPopupMenu(self), e.GetPosition())

def main():

    ex = wx.App()
    Example(None)
    ex.MainLoop()


if __name__ == '__main__':
    main()
```

En el ejemplo, creamos un menú contextual para la ventana principal. Este tiene dos items. Uno podrá minimizar la aplicación, el otro podrá terminarla.

```
class MyPopupMenu(wx.Menu):

    def __init__(self, parent):
        super(MyPopupMenu, self).__init__()
```

creamos una clase wx.Menu separada.

```
mmi = wx.MenuItem(self, wx.NewId(), 'Minimize')
self.AppendItem(mmi)
self.Bind(wx.EVT_MENU, self.OnMinimize, mmi)
```

Un item de menú es creado. Anexado al Menú contextual. Incluso un manejador es enganchado a éste item de menú.

```
self.Bind(wx.EVT_RIGHT_DOWN, self.OnRightDown)
```

Si hacemos clic derecho sobre el marco, llamamos el método OnRightDown(). Para esto, usamos el enlazador wx.EVT_RIGHT_DOWN.

```
def OnRightDown(self, e):
    self.PopupMenu(MyPopupMenu(self), e.GetPosition())
```

En el método OnRightDown(), llamamos el método PopupMenu(). Este método muestra el Menú contextual. El primer parámetro es el menu a ser mostrado. El segundo parámetro es la posición, donde el Menú contextual aparece. Los menúes contextuales aparecen en el punto del cursor del ratón. Para obtener la posición actual del ratón, llamamos el método GetPosition() del objeto suministrado.



Figura: Menú contextual

Barra de Herramientas

Los menús agrupan todos los comandos que podemos usar en una Aplicación. Las Barras de Herramientas proveen a acceso rápido a los comandos usados mas frecuentemente.

para crear una barra de Herramientas, llamamos el método CreateToolBar() del componente Marcos.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
'''
ZetCode wxPython tutorial
Este ejemplo crea una barra de herramientas simple.
author: Jan Bodnar
website: www.zetcode.com
last modified: September 2011
'''
```

```
import wx

class Example(wx.Frame):

    def __init__(self, *args, **kwargs):
        super(Example, self).__init__(*args, **kwargs)

        self.InitUI()

    def InitUI(self):

        toolbar = self.CreateToolBar()
        qtool = toolbar.AddLabelTool(wx.ID_ANY, 'Quit',
wx.Bitmap('texit.png'))
        toolbar.Realize()

        self.Bind(wx.EVT_TOOL, self.OnQuit, qtool)

        self.SetSize((250, 200))
        self.SetTitle('Simple toolbar')
        self.Centre()
        self.Show(True)

    def OnQuit(self, e):
        self.Close()

def main():

    ex = wx.App()
    Example(None)
    ex.MainLoop()


if __name__ == '__main__':
    main()
```

En nuestro ejemplo, tenemos una barra de Herramientas con una herramienta. La herramienta cerrará la aplicación, cuando hagamos clic sobre ella..

```
toolbar = self.CreateToolBar()
```

Creamos una barra de Herramientas. Por omisión, la barra de herramientas es horizontal, no tiene bordes y iconos mostrables.

```
qtool = toolbar.AddLabelTool(wx.ID_ANY, 'Quit', wx.Bitmap('texit.png'))
```

Para crear una herramienta Barra de Herramientas, llamamos el método AddLabelTool(). El segundo parámetro es la etiqueta de la herramienta, el tercero es la imagen de la herramienta. Note que la etiqueta no es visible, porque el estilo por omisión muestra solo iconos.

```
toolbar.Realize()
```

Después que ponemos nuestras items en la barra de herramientas, llamamos al método Realize(). Llamando este método no es obligatorio sobre Linux pero sí sobre Windows.

```
self.Bind(wx.EVT_TOOL, self.OnQuit, qtool)
```

Para hacer frente a eventos de la barra de herramientas, usamos el enlazador wx.EVT_TOOL.

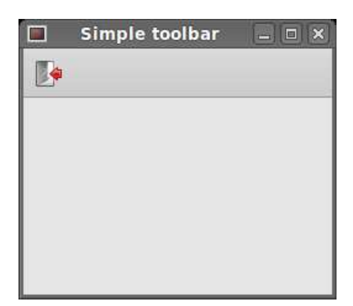


Figura: Barra de herramientas simple

Si buscamos crear más que una barra de herramientas, debemos hacerlo diferenciadamente.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
'''
ZetCode wxPython tutorial
En este ejemplo, creamos dos Barras de Herramientas horizontales.
author: Jan Bodnar
website: www.zetcode.com
last modified: September 2011
'''

import wx

class Example(wx.Frame):

    def __init__(self, *args, **kwargs):
        super(Example, self).__init__(*args, **kwargs)

        self.InitUI()

    def InitUI(self):

        vbox = wx.BoxSizer(wx.VERTICAL)

        toolbar1 = wx.ToolBar(self)
        toolbar1.AddLabelTool(wx.ID_ANY, '', wx.Bitmap('tnew.png'))
        toolbar1.AddLabelTool(wx.ID_ANY, '', wx.Bitmap('topen.png'))
        toolbar1.AddLabelTool(wx.ID_ANY, '', wx.Bitmap('tsave.png'))
        toolbar1.Realize()

        toolbar2 = wx.ToolBar(self)
        qtool = toolbar2.AddLabelTool(wx.ID_EXIT, '',
wx.Bitmap('texit.png'))
        toolbar2.Realize()

        vbox.Add(toolbar1, 0, wx.EXPAND)
        vbox.Add(toolbar2, 0, wx.EXPAND)

        self.Bind(wx.EVT_TOOL, self.OnQuit, qtool)

        self.SetSizer(vbox)
```

```
        self.SetSize((300, 250))
        self.SetTitle('Toolbars')
        self.Centre()
        self.Show(True)

    def OnQuit(self, e):
        self.Close()

def main():

    ex = wx.App()
    Example(None)
    ex.MainLoop()


if __name__ == '__main__':
    main()
```

En el ejemplo de arriba, creamos dos barras de herramientas horizontales.

```
toolbar1 = wx.ToolBar(self)
...
toolbar2 = wx.ToolBar(self)
```

creamos dos objetos barra de herramientas. Y los ponemos dentro de una caja vertical.

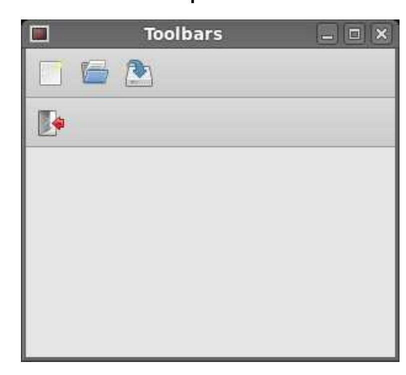


Figura: Barras de Herramientas

Activar, desactivar

En el siguiente ejemplo, mostraremos, como podemos activar y desactivar botones de una barra de herramientas. Veremos también una línea separadora.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
'''
ZetCode wxPython tutorial
En este ejemplo, creamos dos horizontal
Barra de Herramientas.
author: Jan Bodnar
website: www.zetcode.com
last modified: September 2011
'''
import wx
class Example(wx.Frame):
```

```
    def __init__(self, *args, **kwargs):
        super(Example, self).__init__(*args, **kwargs)

        self.InitUI()

    def InitUI(self):

        self.count = 5

        self.toolbar = self.CreateToolBar()
        tundo = self.toolbar.AddLabelTool(wx.ID_UNDO, '',
wx.Bitmap('tundo.png'))
        tredo = self.toolbar.AddLabelTool(wx.ID_REDO, '',
wx.Bitmap('tredo.png'))
        self.toolbar.EnableTool(wx.ID_REDO, False)
        self.toolbar.AddSeparator()
        texit = self.toolbar.AddLabelTool(wx.ID_EXIT, '',
wx.Bitmap('texit.png'))
        self.toolbar.Realize()

        self.Bind(wx.EVT_TOOL, self.OnQuit, texit)
        self.Bind(wx.EVT_TOOL, self.OnUndo, tundo)
        self.Bind(wx.EVT_TOOL, self.OnRedo, tredo)

        self.SetSize((250, 200))
        self.SetTitle('Undo redo')
        self.Centre()
        self.Show(True)

    def OnUndo(self, e):
        if self.count > 1 and self.count <= 5:
            self.count = self.count - 1

        if self.count == 1:
            self.toolbar.EnableTool(wx.ID_UNDO, False)

        if self.count == 4:
            self.toolbar.EnableTool(wx.ID_REDO, True)

    def OnRedo(self, e):
        if self.count < 5 and self.count >= 1:
            self.count = self.count + 1

        if self.count == 5:
            self.toolbar.EnableTool(wx.ID_REDO, False)

        if self.count == 2:
            self.toolbar.EnableTool(wx.ID_UNDO, True)


    def OnQuit(self, e):
        self.Close()

def main():
```

```
    ex = wx.App()
    Example(None)
    ex.MainLoop()


if __name__ == '__main__':
    main()
```

En nuestro ejemplo, tenemos tres botones de la barra de herramientas. Un botón es para salir de las aplicaciones. Los otros dos son los botones deshacer y rehacer. Ellos simulan la funcionalidad deshacer/rehacer en una Aplicación. (Para un ejemplo real, ver consejos y trucos) tenemos 4 cambios. Los botones deshacer y rehacer son deshabilitados en forma acorde.

```
self.toolbar.EnableTool(wx.ID_REDO, False)
self.toolbar.AddSeparator()
```

Al comienzo, el botón rehacer está deshabilitado. Al llamar al método EnableTool(). Podemos crear algúnos grupos lógicos dentro de una barra de Herramientas. Podemos separar varios grupos de botones por una línea vertical pequeña. Para hacer esto, llamamos al método AddSeparator().

```
def OnUndo(self, e):
    if self.count > 1 and self.count <= 5:
        self.count = self.count - 1

    if self.count == 1:
        self.toolbar.EnableTool(wx.ID_UNDO, False)

    if self.count == 4:
        self.toolbar.EnableTool(wx.ID_REDO, True)
```

Simulamos la funcionalidad deshacer y rehacer. Tenemos 4 cambios. Si no hay nada para deshacer, el botón deshacer está deshabilitado. Después de deshacer el primer cambio, habilitamos el botón rehacer. La misma lógica se aplica para el método OnRedo().

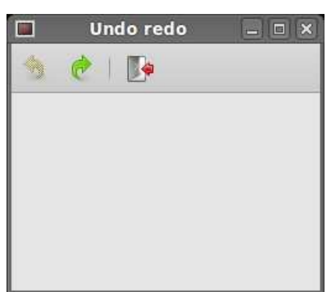


Figura: Undo rehacer

En esta parte del tutorial de wxPython, hemos trabajado con menús y Barras de Herramientas.

Gestión de Diseño en wxPython

Una aplicación típica consiste de varios componentes. Estos componentes son colocados en el interior de componentes contenedores. Un programador debe gestionar el diseño de las aplicaciones. Esta no es una tarea fácil. En wxPython tenemos dos opciones.

- Posicionamiento Absoluto
- Dimensionadores (Sizers)

Posicionamiento Absoluto

El programador especifica la posición y el tamaño de cada componente en píxeles. Cuando usted usa Posicionamiento Absoluto, usted tiene que entender varias cosas.

- el tamaño y la posición de un componente no cambia, si usted redimensiona una ventana
- las aplicaciones se ven diferentes sobre distintas plataformas
- cambiando los tipos de letra en vuestras aplicaciones podría estropear el diseño
- si usted decide cambiar vuestro diseño, usted debe rehacer vuestro diseño completamente, la cual es tedioso y lleva tiempo

Puede haber situaciones, donde podemos posiblemente usar Posicionamiento Absoluto. Por ejemplo, mis tutoriales. Yo no busco hacer los ejemplos muy difíciles, asi que a menudo usamos Posicionamiento Absoluto para explicar un tema. Pero sobre todo, en programas de la vida real, los programadores usan Dimensionadores.

En nuestro ejemplo tenemos un esqueleto simple de un editor de texto. Si redimensionamos la ventana, el tamaño de salida wx.TextCtrl no cambiará como esperaríamos.

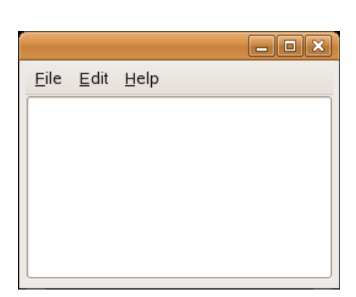


Figura: antes del redimensionamiento

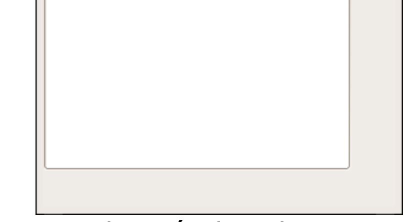


Figura: después de redimensionamiento

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# absolute.py

import wx

class Example(wx.Frame):

    def __init__(self, parent, title):
        super(Example, self).__init__(parent, title=title,
            size=(260, 180))

        self.InitUI()
        self.Centre()
        self.Show()

    def InitUI(self):
```

```
        panel = wx.Panel(self, -1)

        menubar = wx.MenuBar()
        filem = wx.Menu()
        editm = wx.Menu()
        helpm = wx.Menu()

        menubar.Append(filem, '&File')
        menubar.Append(editm, '&Edit')
        menubar.Append(helpm, '&Help')
        self.SetMenuBar(menubar)

        wx.TextCtrl(panel, pos=(3, 3), size=(250, 150))


if __name__ == '__main__':

    app = wx.App()
    Example(None, title='')
    app.MainLoop()
```

En el ejemplo de arriba, posicionamos un control de texto en coordenadas absolutas.

```
wx.TextCtrl(panel, pos=(3, 3), size=(250, 150))
```

Hacemos el Posicionamiento Absoluto en el constructor del wx.TextCtrl. En nuestro caso, posicionamos el wx.TextCtrl at x=3, y=3. El ancho es 250px y el alto 150px.

Usando Dimensionadores

Los Dimensionadores tienen en cuenta todos estos problemas, que mencionamos por Posicionamiento Absoluto. Podemos elegir entre estos Dimensionadores:

- wx.BoxSizer
- wx.StaticBoxSizer
- wx.GridSizer
- wx.FlexGridSizer
- wx.GridBagSizer

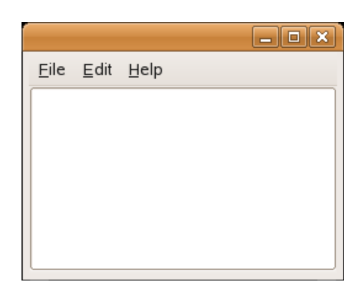



Figura: antes del redimensionamiento

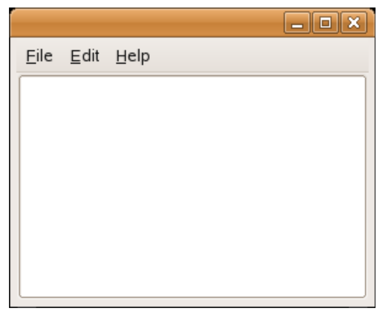


Figura: después del redimensionamiento

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# dimensionador.py
```

```
import wx

class Example(wx.Frame):
    def __init__(self, parent, title):
        super(Example, self).__init__(parent, title=title,
            size=(260, 180))

        self.InitUI()
        self.Centre()
        self.Show()

    def InitUI(self):

        menubar = wx.MenuBar()
        filem = wx.Menu()
        editm = wx.Menu()
        helpm = wx.Menu()

        menubar.Append(filem, '&File')
        menubar.Append(editm, '&Edit')
        menubar.Append(helpm, '&Help')
        self.SetMenuBar(menubar)

        wx.TextCtrl(self)


if __name__ == '__main__':

    app = wx.App()
    Example(None, title='')
    app.MainLoop()
```

En este ejemplo, esta dimensionador no está visible. Colocamos un componente wx.TextCtrl dentro del wx.Frame. El componente wx.Frame tiene un dimensionador especial incorporado. Podemos poner solamente un componente dentro del contenedor wx.Frame. El componente hijo ocupa todo el espacio, lo cual no se le da a los bordes, menu, barra de herramientas y la barra de estado.

wx.BoxSizer

Este dimensionador nos habilita para poner varios componentes dentro de una fila o una columna. Podemos poner otro dimensionador dentro de un dimensionador existente. De esta manera podemos crear diseños muy complejos.

```
 box = wx.BoxSizer(integer orient)
 box.Add(wx.Window window, integer proportion=0, integer flag = 0, integer
border = 0)
```

La orientación pueden ser wx.VERTICAL o wx.HORIZONTAL. El agregado de componentes dentro del wx.BoxSizer se hace por medio del método Add(). Para entenderlo, necesitamos mirar a sus parámetros.

El parámetro proporción definir la razón de como se cambiara la orientación definida de los componentes. Supongamos que tenemos tres botones con las proporciones 0, 1, y 2. Aquellos son agregados dentro de un wx.BoxSizer horizontal. El botón con proporción 0 no cambiará en ningún caso. El botón con proporción 2 cambiará dos veces más que el que tiene proporción 1 en la dimension

horizontal.

Con el parámetro bandera usted puede seguir configurando el comportamiento de los componentes dentro de un wx.BoxSizer. Podemos controlar el borde entre los componentes. Agregaremos algo de espacio entre los componentes en píxeles. Para aplicar bordes necesitamos definir lados, donde el borde podrá usarse. Podemos combinar ellos con el | operator. p.e. wx.LEFT | wx.BOTTOM. Podemos elegir entre estas banderas:

- wx.LEFT
- wx.RIGHT
- wx.BOTTOM
- wx.TOP
- wx.ALL

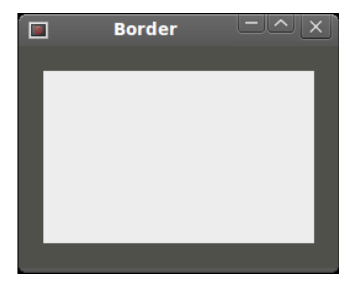


Figura: borde alrededor de un panel

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

# border.py

import wx

class Example(wx.Frame):

    def __init__(self, parent, title):
        super(Example, self).__init__(parent, title=title,
            size=(260, 180))

        self.InitUI()
        self.Centre()
        self.Show()

    def InitUI(self):

        panel = wx.Panel(self)

        panel.SetBackgroundColour('#4f5049')
        vbox = wx.BoxSizer(wx.VERTICAL)

        midPan = wx.Panel(panel)
        midPan.SetBackgroundColour('#ededed')

        vbox.Add(midPan, 1, wx.EXPAND | wx.ALL, 20)
        panel.SetSizer(vbox)


if __name__ == '__main__':
```

```
    app = wx.App()
    Example(None, title='Border')
    app.MainLoop()
```

En el ejemplo de arriba, colocamos algunos espacios alrededor del panel.

```
vbox.Add(midPan, 1, wx.EXPAND | wx.ALL, 20)
```

En border.py tenemos colocado un borde de 20 px alrededor del panel midPan. wx.ALL aplica el tamaño del borde a todos los cuatro lados.

Si usamos la bandera wx.EXPAND, nuestro componente ocupará todo el espacio que se le había asignado. Por último, podemos también definir la alineación de nuestros componentes. Lo hacemos con las banderas siguientes:

- wx.ALIGN_LEFT
- wx.ALIGN_RIGHT
- wx.ALIGN_TOP
- wx.ALIGN_BOTTOM
- wx.ALIGN_CENTER_VERTICAL
- wx.ALIGN_CENTER_HORIZONTAL
- wx.ALIGN_CENTER

Ir a Clase

En el siguientes ejemplo introducimos varias ideas importantes.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# gotoclass.py

import wx

class Example(wx.Frame):

    def __init__(self, parent, title):
        super(Example, self).__init__(parent, title=title,
            size=(390, 350))

        self.InitUI()
        self.Centre()
        self.Show()

    def InitUI(self):

        panel = wx.Panel(self)

        font = wx.SystemSettings_GetFont(wx.SYS_SYSTEM_FONT)
        font.SetPointSize(9)

        vbox = wx.BoxSizer(wx.VERTICAL)

        hbox1 = wx.BoxSizer(wx.HORIZONTAL)
        st1 = wx.StaticText(panel, label='Class Name')
        st1.SetFont(font)
        hbox1.Add(st1, flag=wx.RIGHT, border=8)
        tc = wx.TextCtrl(panel)
```

```
        hbox1.Add(tc, proportion=1)
        vbox.Add(hbox1, flag=wx.EXPAND|wx.LEFT|wx.RIGHT|wx.TOP, border=10)

        vbox.Add((-1, 10))

        hbox2 = wx.BoxSizer(wx.HORIZONTAL)
        st2 = wx.StaticText(panel, label='Matching clases')
        st2.SetFont(font)
        hbox2.Add(st2)
        vbox.Add(hbox2, flag=wx.LEFT | wx.TOP, border=10)

        vbox.Add((-1, 10))

        hbox3 = wx.BoxSizer(wx.HORIZONTAL)
        tc2 = wx.TextCtrl(panel, style=wx.TE_MULTILINE)
        hbox3.Add(tc2, proportion=1, flag=wx.EXPAND)
        vbox.Add(hbox3, proportion=1, flag=wx.LEFT|wx.RIGHT|wx.EXPAND,
            border=10)

        vbox.Add((-1, 25))

        hbox4 = wx.BoxSizer(wx.HORIZONTAL)
        cb1 = wx.CheckBox(panel, label='Case Sensitive')
        cb1.SetFont(font)
        hbox4.Add(cb1)
        cb2 = wx.CheckBox(panel, label='Nested class')
        cb2.SetFont(font)
        hbox4.Add(cb2, flag=wx.LEFT, border=10)
        cb3 = wx.CheckBox(panel, label='Non-Project class')
        cb3.SetFont(font)
        hbox4.Add(cb3, flag=wx.LEFT, border=10)
        vbox.Add(hbox4, flag=wx.LEFT, border=10)

        vbox.Add((-1, 25))

        hbox5 = wx.BoxSizer(wx.HORIZONTAL)
        btn1 = wx.button(panel, label='Ok', size=(70, 30))
        hbox5.Add(btn1)
        btn2 = wx.button(panel, label='Close', size=(70, 30))
        hbox5.Add(btn2, flag=wx.LEFT|wx.BOTTOM, border=5)
        vbox.Add(hbox5, flag=wx.ALIGN_RIGHT|wx.RIGHT, border=10)

        panel.SetSizer(vbox)


if __name__ == '__main__':

    app = wx.App()
    Example(None, title='Ir a Clase')
    app.MainLoop()
```

El diseño es sencillo. Creamos un dimensionador vertical. Ponemos entonces cinco dimensionadores horizontales dentro de él.

```
font = wx.SystemSettings_GetFont(wx.SYS_SYSTEM_FONT)
font.SetPointSize(9)
```

El tipo de letra por omisión del sistema era 10px. En mi plataforma, éste fue demasiado grande para este tipo de ventana. Asi que puse el tamaño de la fuente a 9px.

```
vbox.Add(hbox3, proportion=1, flag=wx.LEFT|wx.RIGHT|wx.EXPAND, border=10)

vbox.Add((-1, 25))
```

Ya sabemos que podemos controlar la distancia entre componentes combinando el parámetro bandera con el parámetro borde. Pero esta es una restricción real. En el método Add() podemos especificar solamente un borde para todos los lados dados. En nuestro ejemplo, damos 10px a la derecha y a la izquierda. Pero Nosotros no podemos dar 25 px para el botón. Que podemos hacer para dar 10px para el botón, o 0px. Si omitimos wx.BOTTOM. Así que si necesitamos diferentes valores, podemos agregar algunos espacios extras. Con el método Add(), podemos insertar componentes y espacio también.

```
vbox.Add(hbox5, flag=wx.ALIGN_RIGHT|wx.RIGHT, border=10)
```

Colocamos los dos botones sobre el lado derecho de la ventana. ¿Como lo hacemos? tres cosas son importantes para lograr esto. La proporción, la bandera de alineación y la bandera wx.EXPAND. La proporción debe ser cero. Los botones no deberían cambiar su tamaño, cuando redimensionamos nuestras ventanas. No debemos especificar la bandera wx.EXPAND. Los botones ocupan solamente el área que tienen asignada para ellos. Y finalmente, debemos especificar la bandera wx.ALIGN_RIGHT. El dimensionador horizontal se extiende desde el lado izquierdo de la ventana al lado derecho. Por lo que si especificamos la bandera wx.ALIGN_RIGHT, los botones son colocados al lado derecho. Exactamente, como queríamos.

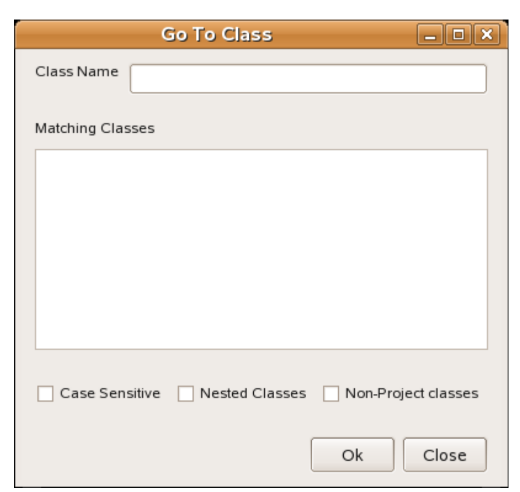


Figura: A Ir a clase ventana

wx.GridSizer

El wx.GridSizer establece los componentes en una tabla bidimensional. Cada celda dentro de la tabla tiene el mismo tamaño.

```
wx.GridSizer(int rows=1, int cols=0, int vgap=0, int hgap=0)
```

En el constructor especificamos el número de filas y columnas en la tabla. Y el espacio vertical y horizontal entre nuestras celdas.

En nuestro ejemplo creamos un esqueleto de una calculadora. Esto es un ejemplo perfecto para wx.GridSizer.

```
#!/usr/bin/python
```

```
# -*- coding: utf-8 -*-
# calculator.py

import wx
class Example(wx.Frame):

    def __init__(self, parent, title):
        super(Example, self).__init__(parent, title=title,
            size=(300, 250))

        self.InitUI()
        self.Centre()
        self.Show()

    def InitUI(self):

        menubar = wx.MenuBar()
        fileMenu = wx.Menu()
        menubar.Append(fileMenu, '&File')
        self.SetMenuBar(menubar)


        vbox = wx.BoxSizer(wx.VERTICAL)
        self.display = wx.TextCtrl(self, style=wx.TE_RIGHT)
        vbox.Add(self.display, flag=wx.EXPAND|wx.TOP|wx.BOTTOM, border=4)
        gs = wx.GridSizer(4, 4, 5, 5)

        gs.AddMany( [(wx.button(self, label='Cls'), 0, wx.EXPAND),
            (wx.button(self, label='Bck'), 0, wx.EXPAND),
            (wx.StaticText(self), wx.EXPAND),
            (wx.button(self, label='Close'), 0, wx.EXPAND),
            (wx.button(self, label='7'), 0, wx.EXPAND),
            (wx.button(self, label='8'), 0, wx.EXPAND),
            (wx.button(self, label='9'), 0, wx.EXPAND),
            (wx.button(self, label='/'), 0, wx.EXPAND),
            (wx.button(self, label='4'), 0, wx.EXPAND),
            (wx.button(self, label='5'), 0, wx.EXPAND),
            (wx.button(self, label='6'), 0, wx.EXPAND),
            (wx.button(self, label='*'), 0, wx.EXPAND),
            (wx.button(self, label='1'), 0, wx.EXPAND),
            (wx.button(self, label='2'), 0, wx.EXPAND),
            (wx.button(self, label='3'), 0, wx.EXPAND),
            (wx.button(self, label='-'), 0, wx.EXPAND),
            (wx.button(self, label='0'), 0, wx.EXPAND),
            (wx.button(self, label='.'), 0, wx.EXPAND),
            (wx.button(self, label='='), 0, wx.EXPAND),
            (wx.button(self, label='+'), 0, wx.EXPAND) ])

        vbox.Add(gs, proportion=1, flag=wx.EXPAND)
        self.SetSizer(vbox)


if __name__ == '__main__':

    app = wx.App()
    Example(None, title='Calculator')
```

```
    app.MainLoop()
```

Observe como nos arreglamos para poner a espacio entre los botones Bck y Close. Simplemente ponemos aquí un wx.StaticText vacio.

En nuestro ejemplo estamos usandoel método AddMany(). Esto es un método conveniente para agregar múltiples componentes a la vez.

```
gs.AddMany( [(wx.button(self, label='Cls'), 0, wx.EXPAND),
...
```

Los componentes son colocados dentro de la tabla en orden, aquellos son agregados donde la primer fila es llenada primero, entonces la segunda fila etc.

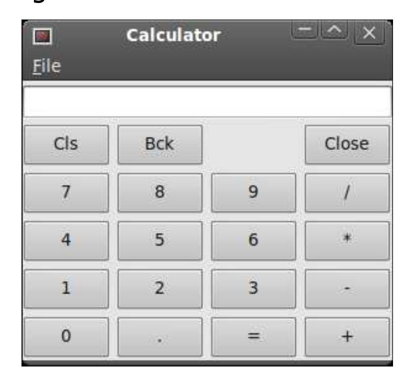


Figura: Calculadora

wx.FlexGridSizer

Este dimensionador es similar a wx.GridSizer. Éste tambien expone sus componentes en una tabla bidimensional. Esto añade cierta flexibilidad para él. Las celdas de wx.GridSizer son del mismo tamaño. Todas las celdas en wx.FlexGridSizer tienen el mismo alto en una fila. Todas las celdas tienen el mismo ancho en una columna Pero todas las filas y columnas no son necesariamente del mismo alto o ancho.

```
 wx.FlexGridSizer(int rows=1, int cols=0, int vgap=0, int hgap=0)
```

rows y *cols* especifican el número de filas y columnas en un dimensionador. *vgap* y *hgap* agregan algo de espacio entre componentes en ambas direcciones.

Muchas veces desarrolladores tienen que desarrollar diálogos para entrada y modificación de datos. Me parece que wx.FlexGridSizer es adecuado para tal tarea. Un desarrollador pueden fácilmente establecer una ventana de diálogo con este dimensionador. Esto es también posible lograr esto con un wx.GridSizer, pero no se vería bien, por la restricción de que cada celda debe tener el mismo tamaño.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# review.py


import wx

class Example(wx.Frame):

    def __init__(self, parent, title):
        super(Example, self).__init__(parent, title=title,
            size=(300, 250))
```

```
        self.InitUI()
        self.Centre()
        self.Show()

    def InitUI(self):

        panel = wx.Panel(self)

        hbox = wx.BoxSizer(wx.HORIZONTAL)

        fgs = wx.FlexGridSizer(3, 2, 9, 25)

        title = wx.StaticText(panel, label="Title")
        author = wx.StaticText(panel, label="Author")
        review = wx.StaticText(panel, label="Review")

        tc1 = wx.TextCtrl(panel)
        tc2 = wx.TextCtrl(panel)
        tc3 = wx.TextCtrl(panel, style=wx.TE_MULTILINE)

        fgs.AddMany([(title), (tc1, 1, wx.EXPAND), (author),
            (tc2, 1, wx.EXPAND), (review, 1, wx.EXPAND), (tc3, 1,
wx.EXPAND)])

        fgs.AddGrowableRow(2, 1)
        fgs.AddGrowableCol(1, 1)

        hbox.Add(fgs, proportion=1, flag=wx.ALL|wx.EXPAND, border=15)
        panel.SetSizer(hbox)


if __name__ == '__main__':

    app = wx.App()
    Example(None, title='Review')
    app.MainLoop()
```

El código del ejemplo de arriba, creamos una ventana de Review con un FlexGridSizer.

```
hbox = wx.BoxSizer(wx.HORIZONTAL)
...
hbox.Add(fgs, proportion=1, flag=wx.ALL|wx.EXPAND, border=15)
```

Creamos una caja dimensionadora horizontal en orden para poner algo de espacio (15px) alrededor de la tabla de componentes.

```
fgs.AddMany([(title), (tc1, 1, wx.EXPAND), (author),
    (tc2, 1, wx.EXPAND), (review, 1, wx.EXPAND), (tc3, 1, wx.EXPAND)])
```

Agregaremos componentes al dimensionador con el método AddMany(). Tanto wx.FlexGridSizer como wx.GridSizer comparten este método.

```
fgs.AddGrowableRow(2, 1)
fgs.AddGrowableCol(1, 1)
```

Hacemos la tercer fila y la segunda columna expandibles. De esta manera dejamos los controles de

texto expandirse, cuando la ventana es redimensionada. Los primeros dos controles de texto crecerán en dirección horizontal, el tercero crecerá en ambas direcciones. No debemos olvidar hacer los componentes expandibles (wx.EXPAND) en orden de hacer que realmente funcionen.



Figura: ejemplo de revisión

wx.GridBagSizer

Es el más complicado dimensionador en wxPython. Muchos programadores encuentran difícil de usar. Este tipo de dimensionador no es típico solamente para wxPython. Podemos encontrarlo en otros kits de herramientas también. Aunque éste es más complicado, éste no es sin duda ciencia de cohetes.

Este dimensionador permite posicionamiento explícito de items. Los items pueden también opcionalmente abarcar más de una fila y/o columna. El wx.GridBagSizer tiene un constructor simple.

```
wx.GridBagSizer(integer vgap, integer hgap)
```

La brecha (gap) vertical y horizontal definen el espacio en píxeles usado entre todos los hijos. Agregaremos items a la grilla con el método Add().

```
Add(self, item, tuple pos, tuple span=wx.DefaultSpan, integer flag=0,
     integer border=0, userData=None)
```

El parámetro ***item*** es un componente que usted inserta dentro de la grilla, el ***pos*** especifica la posición en la grilla virtual. La celda de arriba a la izquierda tiene pos de (0, 0). El ***span*** es lo que abarca opcional del componente. p.e. ***span*** de (3, 2) hace abarar al componente a través de 3 filas y 2 columnas. ***flag*** y ***border*** fueron discutidos antes para wx.BoxSizer. Los items en la grilla pueden cambiar su tamaño o mantener el tamaño por omisión, cuando la ventana es redimensionada. Si usted busca que sus items crezcan o reduscan, usted puede usar el siguientes dos métodos:

```
AddGrowableRow(integer row)
AddGrowableCol(integer col)
```

Ventana renombrar

En nuestro primer ejemplo, vamos a crear una ventana renombrar. Ésta tendrá un wx.StaticText, un wx.TextCtrl y dos wx.button-s.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

# rename.py

import wx
```

```
class Example(wx.Frame):

    def __init__(self, parent, title):
        super(Example, self).__init__(parent, title=title,
            size=(320, 130))

        self.InitUI()
        self.Centre()
        self.Show()

    def InitUI(self):

        panel = wx.Panel(self)
        sizer = wx.GridBagSizer(4, 4)

        text = wx.StaticText(panel, label="Rename To")
        sizer.Add(text, pos=(0, 0), flag=wx.TOP|wx.LEFT|wx.BOTTOM, border=5)

        tc = wx.TextCtrl(panel)
        sizer.Add(tc, pos=(1, 0), span=(1, 5),
            flag=wx.EXPAND|wx.LEFT|wx.RIGHT, border=5)

        buttonOk = wx.button(panel, label="Ok", size=(90, 28))
        buttonClose = wx.button(panel, label="Close", size=(90, 28))
        sizer.Add(buttonOk, pos=(3, 3))
        sizer.Add(buttonClose, pos=(3, 4), flag=wx.RIGHT|wx.BOTTOM,
border=5)

        sizer.AddGrowableCol(1)
        sizer.AddGrowableRow(2)
        panel.SetSizerAndFit(sizer)


if __name__ == '__main__':
    app = wx.App()
    Example(None, title='Rename')
    app.MainLoop()
```

Nosotros debemos ver en la ventana como una gran grilla o tabla.

```
text = wx.StaticText(panel, label="Rename To")
sizer.Add(text, pos=(0, 0), flag=wx.TOP|wx.LEFT|wx.BOTTOM, border=5)
```

El texto "Rename to" va a la esquina superior izquierda. Así especificamos la posición (0, 0). Además añadimos algo de espacio para el botón, izquierdo y de fondo.

```
tc = wx.TextCtrl(panel)
sizer.Add(tc, pos=(1, 0), span=(1, 5),
    flag=wx.EXPAND|wx.LEFT|wx.RIGHT, border=5)
```

El wx.TextCtrl va al principio de la segunda fila (1, 0). Recuerde, que contamos desde cero. Esto expande 1 fila y 5 columnas. (1, 5). Y ponemos 5 píxeles de espacio a la izquierda y a la derecha del componente.

```
sizer.Add(buttonOk, pos=(3, 3))
sizer.Add(buttonClose, pos=(3, 4), flag=wx.RIGHT|wx.BOTTOM, border=5)
```

Ponemos dos botones dentro de la cuarta fila. La tercer fila se deja vacia, de modo que tenemos algo de espacio entre el wx.TextCtrl y los botones. Ponemos el botón OK dentro de la cuarta columna y el botón close dentro de la quinta. Tenga en cuenta que una vez aplicamos algo de espacio a un componente, esto no es aplicado al conjunto de filas. Es por eso que no especificamos espacio al fondo para el botón OK. Un lector atento puede notar, que no especificamos cualquier espacio entre los dos botones. p.e. no incluimos ningún espacio a la derecha del botón OK, o a la derecha del botón Close. En el constructor del wx.GridBagSizer, ponemos algo de espacio entre todos los componentes. Esto es algo de espacio ya.

```
sizer.AddGrowableCol(1)
sizer.AddGrowableRow(2)
```

La última cosa que debemos hacer, es hacer nuestros diálogos redimensionables. Hacemos la segunda columna y la tercer fila expandibles. Ahora podemos expandir o comprimir nuestras ventanas. Intentemos comentar estas dos líneas y vemos que pasa.

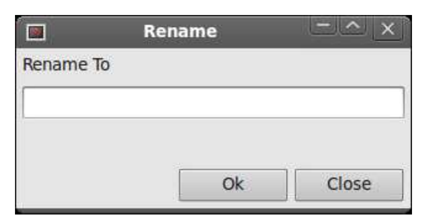


Figura: Ventana renombrar

Ejemplo de nueva clase

En el ejemplo siguiente creamos una ventana, que puede ser encontrada en JDeveloper. Esto es una ventana para crear una nueva clase en Java.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

# newclass.py

import wx

class Example(wx.Frame):

    def __init__(self, parent, title):
        super(Example, self).__init__(parent, title=title,
            size=(450, 350))

        self.InitUI()
        self.Centre()
        self.Show()

    def InitUI(self):

        panel = wx.Panel(self)

        sizer = wx.GridBagSizer(5, 5)

        text1 = wx.StaticText(panel, label="Java Class")
        sizer.Add(text1, pos=(0, 0), flag=wx.TOP|wx.LEFT|wx.BOTTOM,
            border=15)
```

```
        icon = wx.StaticBitmap(panel, bitmap=wx.Bitmap('exec.png'))
        sizer.Add(icon, pos=(0, 4), flag=wx.TOP|wx.RIGHT|wx.ALIGN_RIGHT,
            border=5)

        line = wx.StaticLine(panel)
        sizer.Add(line, pos=(1, 0), span=(1, 5),
            flag=wx.EXPAND|wx.BOTTOM, border=10)

        text2 = wx.StaticText(panel, label="Name")
        sizer.Add(text2, pos=(2, 0), flag=wx.LEFT, border=10)

        tc1 = wx.TextCtrl(panel)
        sizer.Add(tc1, pos=(2, 1), span=(1, 3), flag=wx.TOP|wx.EXPAND)

        text3 = wx.StaticText(panel, label="Package")
        sizer.Add(text3, pos=(3, 0), flag=wx.LEFT|wx.TOP, border=10)

        tc2 = wx.TextCtrl(panel)
        sizer.Add(tc2, pos=(3, 1), span=(1, 3), flag=wx.TOP|wx.EXPAND,
            border=5)

        button1 = wx.button(panel, label="Browse...")
        sizer.Add(button1, pos=(3, 4), flag=wx.TOP|wx.RIGHT, border=5)

        text4 = wx.StaticText(panel, label="Extends")
        sizer.Add(text4, pos=(4, 0), flag=wx.TOP|wx.LEFT, border=10)

        combo = wx.ComboBox(panel)
        sizer.Add(combo, pos=(4, 1), span=(1, 3),
            flag=wx.TOP|wx.EXPAND, border=5)

        button2 = wx.button(panel, label="Browse...")
        sizer.Add(button2, pos=(4, 4), flag=wx.TOP|wx.RIGHT, border=5)

        sb = wx.StaticBox(panel, label="Optional Attributes")

        boxsizer = wx.StaticBoxSizer(sb, wx.VERTICAL)
        boxsizer.Add(wx.CheckBox(panel, label="Public"),
            flag=wx.LEFT|wx.TOP, border=5)
        boxsizer.Add(wx.CheckBox(panel, label="Generate Default
Constructor"),
            flag=wx.LEFT, border=5)
        boxsizer.Add(wx.CheckBox(panel, label="Generate Main method"),
            flag=wx.LEFT|wx.BOTTOM, border=5)
        sizer.Add(boxsizer, pos=(5, 0), span=(1, 5),
            flag=wx.EXPAND|wx.TOP|wx.LEFT|wx.RIGHT , border=10)

        button3 = wx.button(panel, label='Help')
        sizer.Add(button3, pos=(7, 0), flag=wx.LEFT, border=10)

        button4 = wx.button(panel, label="Ok")
        sizer.Add(button4, pos=(7, 3))

        button5 = wx.button(panel, label="Cancel")
        sizer.Add(button5, pos=(7, 4), span=(1, 1),
            flag=wx.BOTTOM|wx.RIGHT, border=5)
```

```
        sizer.AddGrowableCol(2)

        panel.SetSizer(sizer)


if __name__ == '__main__':

    app = wx.App()
    Example(None, title="Create Java Class")
    app.MainLoop()
```

Éste es un diseño más complicado. Usamos tanto un wx.GridBagSizer como un wx.StaticBoxsizer.

```
line = wx.StaticLine(panel)
sizer.Add(line, pos=(1, 0), span=(1, 5),
    flag=wx.EXPAND|wx.BOTTOM, border=10)
```

Éste es una línea que es usada para separar grupos de componentes en el diseño.

```
icon = wx.StaticBitmap(panel, bitmap=wx.Bitmap('exec.png'))
sizer.Add(icon, pos=(0, 4), flag=wx.TOP|wx.RIGHT|wx.ALIGN_RIGHT,
    border=5)
```

Ponemos un wx.StaticBitmap dentro de la primer fila de la grilla, lo colocamos sobre el lado derecho de la fila.

```
sb = wx.StaticBox(panel, label="Optional Attributes")
boxsizer = wx.StaticBoxSizer(sb, wx.VERTICAL)
```

wxStaticBoxSizer es similar a un wx.BoxSizer normal pero este agrega una caja estática alrededor del dimensionador. Ponemos casillas de verificación dentro de la caja estática del dimensionador.

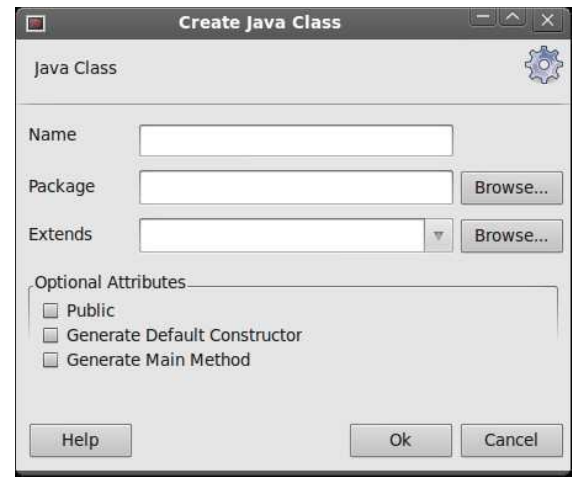


Figura: Ventana nueva clase

---

esta parte del tutorial de wxPython fue dedicada a Gestión de Diseño.

Eventos en wxPython

Eventos son parte integral de cada GUI de la aplicación. Todas las aplicaciones GUI son manejadas por

eventos. Una aplicación reacciona a diferentes tipos los cuales son generados durante su vida. Los eventos son generados principalmente por el usuario de una aplicación. Pero aquellos pueden ser generados por otros medios también. p.e. conexión a Internet, manejador de ventanas, temporizador (timer). Por lo que cuando llamamos al método MainLoop(), nuestras aplicaciones esperan por eventos a ser generados. El método MainLoop() termina cuando salimos de las aplicaciones.

Definiciones

Event es una pieza de a nivel información de aplicaciones desde el subyacente marco de trabajo (framework), tipicamente el kit de herramientas GUI. incluso/aún **bucle** es una construcción de programación que espera por y envia eventos o mensajes en un programa. Aún el bucle repetidamente mira por eventos a procesar. Un despachador es un proceso el cual mapea eventos al **gestionador asincrónico (handler)**. Los gestionadores asincrónicos son métodos que reaccionan a eventos.

Objeto de evento es un objeto asociado con el evento. Éste es usualmente una ventana. Aún el tipo es un evento único, que tiene que ser generado. Aún el **enlazador** es un objeto, que se enlaza a un tipo de gestionador asincrónico.

Un ejemplo aún mas sencillo

En la siguiente sección describiremos un evento simple. Hablaremos acerca de un evento de movimiento.

Un evento movimiento es generado, cuando movemos una ventana a una nueva posición. El tipo de evento es **wx.MoveEvent**. Incluso el enlazador para esto es **wx.EVT_MOVE**

```
#!/usr/bin/python
# moveevent.py

import wx

class MoveEvent(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(250, 180))

        wx.StaticText(self, -1, 'x:', (10,10))
        wx.StaticText(self, -1, 'y:', (10,30))
        self.st1 = wx.StaticText(self, -1, '', (30, 10))
        self.st2 = wx.StaticText(self, -1, '', (30, 30))

        self.Bind(wx.EVT_MOVE, self.OnMove)

        self.Centre()
        self.Show(True)

    def OnMove(self, event):
        x, y = event.GetPosition()
        self.st1.SetLabel(str(x))
        self.st2.SetLabel(str(y))


app = wx.App()
MoveEvent(None, -1, 'move event')
app.MainLoop()
```

El ejemplo muestra la posición actual de la ventana.

```
self.Bind(wx.EVT_MOVE, self.OnMove)
```

Aquí vinculamos el enlazador *wx.EVT_MOVE* al método *OnMove()*.

```
def OnMove(self, event):
    x, y = event.GetPosition()
```

Incluso el parámetro en el método *OnMove()* es un objeto específico para un tipo de evento particular. En nuestro caso éste es la instancia de una clase *wx.MoveEvent*. Este objeto retiene información acerca del evento. Por ejemplo incluso el objecto o la posición de la ventana. En nuestro caso incluso el objecto es el componente wx.Frame. Podemos descubrir la posición actual al llamar el método *GetPosition()* del evento.



Figura: evento movimiento

Enlace de eventos

Trabajar con eventos es sencillo en wxPython. Estos son los tres pasos:

- Identificar el nombre del evento a enlazar: wx.EVT_SIZE, wx.EVT_CLOSE etc
- Crear un gestionador asincrónico del evento. Esto es un método, que es llamado, cuando un evento es generado
- Enlazar un evento a un gestionador asincrónico del evento.

En wxPython decimos enlazar un método a un evento. Algunas veces una palabra gancho es usada. Usted enlaza un evento al llamar el métodoBind(). El método tiene los siguientes parámetros:

```
Bind(event, handler, source=None, id=wx.ID_ANY, id2=wx.ID_ANY)
```

- ***event*** es uno de los objetos EVT_*. Este especifica el tipo del evento.
- ***handler*** es un objeto para ser llamado. En otras palabras, esto es una método, que Un programador binds a un evento.
- Parámetro ***source*** es usado cuando buscamos diferenciar al mismo tipo de evento desde diferentes componentes.
- ***id*** parámetro es usado, cuando tenemos múltiples botones, items de menú etc. El id es usado para diferenciar entre ellos.
- ***id2*** es usado cuando éste es deseable enlazar un manejador a un rango de ids, tales como EVT_MENU_RANGE.

Note que el método Bind() es definido en la clase EvtHandler. Éste es la clase, desde la cual hereda wx.Window. wx.Window es una clase base para la mayoría de los componentes en wxPython. Ésta es también un proceso inverso. Si buscamos desenlazar un método desde un evento, llamamos el método Unbind(). Éste tiene los mismos parámetros como el de arriba.

Vetando eventos

Algunas veces necesitamos parar el procesamiento de un evento. Para hacer esto, llamamos el método *Veto()*.

```
#!/usr/bin/python
# veto.py

import wx

class Veto(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(250, 200))
        self.Bind(wx.EVT_CLOSE, self.OnClose)
        self.Centre()
        self.Show(True)

    def OnClose(self, event):
        dial = wx.MessageDialog(None, 'Are you sure to quit?', 'Question',
            wx.YES_NO | wx.NO_DEFAULT | wx.ICON_QUESTION)
        ret = dial.ShowModal()
        if ret == wx.ID_YES:
            self.Destroy()
        else:
            event.Veto()


app = wx.App()
Veto(None, -1, 'Veto')
app.MainLoop()
```

En nuestro ejemplo, procesamos un *wx.CloseEvent*. Este evento es llamado, cuando hacemos clic en el botón X sobre la barra de títulos, pulsamos Alt+F4 o seleccionamos close desde el menú del sistema. En muchas aplicaciones, buscamos prevenir de cerrar accidentalmente la ventana, si hacemos algunos cambios. Para hacer esto, debemos enlazar el evento *wx.EVT_CLOSE* al enlazador.

```
dial = wx.MessageDialog(None, 'Are you sure to quit?', 'Question',
    wx.YES_NO | wx.NO_DEFAULT | wx.ICON_QUESTION)
ret = dial.ShowModal()
```

Durante el evenpara cerrar, Mostraremos un mensaje en un Cuadro de Diálogo.

```
 if ret == wx.ID_YES:
     self.Destroy()
 else:
     event.Veto()
```

Dependiendo del valor retornado, destruimos la ventana, o vetamos el evento. Tenga en cuenta que para cerrar la ventana, deberemos llamar al método *Destroy()*. Al llamar el método *Close()*, terminaríamos en un ciclo sin fin.

Propagación de Eventos

Estos son dos tipos de eventos. Eventos basicos y eventos comandos. Ellos difieren en la propagación. La propagación de eventos es el viaje de los eventos desde componentes hijos a los componentes padres y componentes abuelos etc. Los eventos basicos no se propagan. Los eventos comandos se propagan. Por ejemplo *wx.CloseEvent* es es un evento básico. No tiene sentido que este evento se propague a los componentes padres.

Por omisión, el evento que es atrapado en un manejador de evento detiene la propagación. Para continuar la propagación, deberemos llamar al método *Skip()*.

```
#!/usr/bin/python
# propagate.py

import wx


class MyPanel(wx.Panel):
    def __init__(self, parent, id):
        wx.Panel.__init__(self, parent, id)

        self.Bind(wx.EVT_BUTTON, self.OnClicked)

    def OnClicked(self, event):
        print 'event reached panel class'
        event.Skip()


class MyButton(wx.button):
    def __init__(self, parent, id, label, pos):
        wx.button.__init__(self, parent, id, label, pos)

        self.Bind(wx.EVT_BUTTON, self.OnClicked)

    def OnClicked(self, event):
        print 'event reached botón class'
        event.Skip()


class Propagate(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(250, 150))

        panel = MyPanel(self, -1)

        MyButton(panel, -1, 'Ok', (15, 15))

        self.Bind(wx.EVT_BUTTON, self.OnClicked)

        self.Centre()
        self.Show(True)

    def OnClicked(self, event):
        print 'event reached frame class'
        event.Skip()


app = wx.App()
Propagate(None, -1, 'Propagate')
app.MainLoop()
```

En nuestro ejemplo, tenemos un botón sobre un panel. El panel es colocado en un componente marco. Definimos un manejador para todos los componentes.

Evento alcanzando clase botón
Evento alcanzando clase panel
Evento alcanzando clase marco

Tomamos éste, cuando hacemos clic sobre el botón. El evento viaja desde el botón al panel y al marco.

Intentemos omitir algunos métodos Skip() y vemos, que pasa.

Identificadores de ventanas

Los identificadores de ventanas son enteros que unicamente determinan la identidad de la ventana en el sistema de evento. Estos son tres caminos para crear id's de ventana.

- Hacer que el sistema automáticamente cree un id
- usar indentificadores estándares
- crear su propio id

Cada componente tiene un parámetro id. Éste es un número único en el sistema de eventos. Si trabajamos con múltiples componentes, debemos diferenciarlos entre ellos.

wx.button(parent, -1)
wx.button(parent, wx.ID_ANY)

Si proveemos -1 o wx.ID_ANY para el parámetro id, dejamos que el wxPython cree automáticamente un id para nosotros. Los id's creados automáticamente son siempre negativos, mientras que los id's especificados por el usuario deben siempre ser positivos. Usualmente usamos esta opción cuando no necesitamos cambiar el estado del componente. Por ejemplo un texto estático, que nunca sería cambiado durante la vida de la aplicación. Podemos aún asi obtener el id, si eso buscamos. Éste es un Método *GetId()*, que va a determinar el id para nosotros.

```
#!/usr/bin/python
# automaticids.py

import wx

class AuIds(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(170, 100))

        panel = wx.Panel(self, -1)
        exit = wx.button(panel, -1, 'Exit', (10, 10))

        self.Bind(wx.EVT_BUTTON,  self.OnExit, id=exit.GetId())

        self.Centre()
        self.Show(True)

    def OnExit(self, event):
        self.Close()


app = wx.App()
AuIds(None, -1, '')
app.MainLoop()
```

En este ejemplo, no nos importa acerca del valor actual del id.

```
self.Bind(wx.EVT_BUTTON,  self.OnExit, id=exit.GetId())
```

tomamos el id generado automáticamente al llamar el método *GetId()*.

Indentificadores estándares deberían ser usado siempre y cuando posible. Los indentificadores pueden proveer algunos gráficos estándares o comportamiento sobre algunas plataformas de SO.

```
#!/usr/bin/python
# identifiers.py

import wx

class Identifiers(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(200, 150))

        panel = wx.Panel(self, -1)
        grid = wx.GridSizer(3, 2)

        grid.AddMany([(wx.button(panel, wx.ID_CANCEL), 0, wx.TOP | wx.LEFT, 
9),
            (wx.button(panel, wx.ID_DELETE), 0, wx.TOP, 9),
            (wx.button(panel, wx.ID_SAVE), 0, wx.LEFT, 9),
            (wx.button(panel, wx.ID_EXIT)),
            (wx.button(panel, wx.ID_STOP), 0, wx.LEFT, 9),
            (wx.button(panel, wx.ID_NEW))])


        self.Bind(wx.EVT_BUTTON, self.OnQuit, id=wx.ID_EXIT)

        panel.SetSizer(grid)
        self.Centre()
        self.Show(True)

    def OnQuit(self, event):
        self.Close()

app = wx.App()
Identifiers(None, -1, '')
app.MainLoop()
```

En nuestro ejemplo usamos indentificadores estándares para los botones. Sobre Linux, los botones tienen iconos pequeños.



Figura: indentificadores estándares

La última opción es para usar identificadores nuestros. Definimos nuestros ids globales propios.

Eventos variados

Evento Focus

El foco indica el componente seleccionado actualmente en la aplicación. El texto entrado desde el teclado o pegado desde el portapapeles es enviado al componente, el cual tiene el foco. Estos son dos tipos de eventos concernientes al focus. El evento **wx.EVT_SET_FOCUS**, el cual es generado cuando un componente recibe el foco. El **wx.EVT_KILL_FOCUS** es generado, cuando el componente pierde el foco. El foco es cambiado haciendo clic o por una tecla del teclado. Usualmente Tab/Shift+Tab.

```
#!/usr/bin/python
# focusevent.py

import wx


class MyWindow(wx.Panel):
    def __init__(self, parent):
        wx.Panel.__init__(self, parent, -1)

        self.color = '#b3b3b3'

        self.Bind(wx.EVT_PAINT, self.OnPaint)
        self.Bind(wx.EVT_SIZE, self.OnSize)
        self.Bind(wx.EVT_SET_FOCUS, self.OnSetFocus)
        self.Bind(wx.EVT_KILL_FOCUS, self.OnKillFocus)

    def OnPaint(self, event):
        dc = wx.PaintDC(self)

        dc.SetPen(wx.Pen(self.color))
        x, y = self.GetSize()
        dc.DrawRectangle(0, 0, x, y)

    def OnSize(self, event):
        self.Refresh()

    def OnSetFocus(self, event):
        self.color = '#0099f7'
        self.Refresh()

    def OnKillFocus(self, event):
        self.color = '#b3b3b3'
        self.Refresh()

class FocusEvent(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(350, 250))

        grid = wx.GridSizer(2, 2, 10, 10)
        grid.AddMany([(MyWindow(self), 1, wx.EXPAND|wx.TOP|wx.LEFT,9),
            (MyWindow(self), 1, wx.EXPAND|wx.TOP|wx.RIGHT, 9),
            (MyWindow(self), 1, wx.EXPAND|wx.BOTTOM|wx.LEFT, 9),
            (MyWindow(self), 1, wx.EXPAND|wx.BOTTOM|wx.RIGHT, 9)])
```

```
        self.SetSizer(grid)
        self.Centre()
        self.Show(True)

app = wx.App()
FocusEvent(None, -1, 'focus event')
app.MainLoop()
```

En nuestro ejemplo, tenemos cuatro paneles. El panel con el foco es destacado.

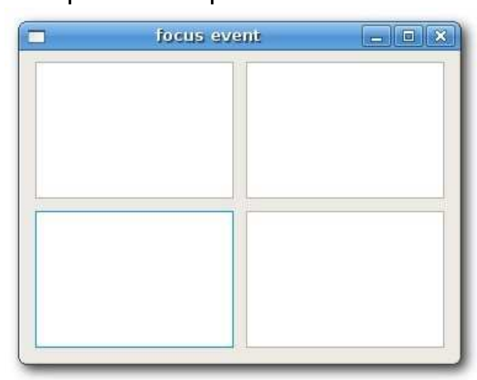


Figura: evento focus

Evento de desplazamiento

El código siguiente es un ejemplo de un wx.ScrollWinEvent. Este evento es generado, cuando hacemos clic sobre una en Barra de desplazamiento construida. Barra de desplazamiento incorporada es desactivada con la llamada al método SetScrollbar(). Para Barras de desplazamiento autónomas, ésta es otro tipo de evento, Es decir, wx.ScrollEvent.

```
#!/usr/bin/python
# myscrollwinevent.py

import wx

class ScrollWinEvent(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title)
        panel = wx.Panel(self, -1)
        self.st = wx.StaticText(panel, -1, '0', (30,0))
        panel.Bind(wx.EVT_SCROLLWIN, self.OnScroll)
        panel.SetScrollbar(wx.VERTICAL, 0, 6, 50);
        self.Centre()
        self.Show(True)

    def OnScroll(self, evt):
        y = evt.GetPosition()
        self.st.SetLabel(str(y))

app = wx.App()
ScrollWinEvent(None, -1, 'scrollwinevent.py')
app.MainLoop()
```

Evento Size

Un wx.SizeEvent es generado, cuando nuestras ventanas son redimensionadas. En nuestro ejemplo, Mostraremos el tamaño de la ventana en la barra de títulos.

```
#!/usr/bin/python
# sizeevent.py

import wx

class SizeEvent(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title)

        self.Bind(wx.EVT_SIZE, self.OnSize)
        self.Centre()
        self.Show(True)

    def OnSize(self, event):
        self.SetTitle(str(event.GetSize()))


app = wx.App()
SizeEvent(None, 1, 'sizeevent.py')
app.MainLoop()
```

```
 self.SetTitle(str(event.GetSize()))
```

Para obtener el tamaño actual de la ventana, llamamos el método *GetSize()* del evento del objeto.

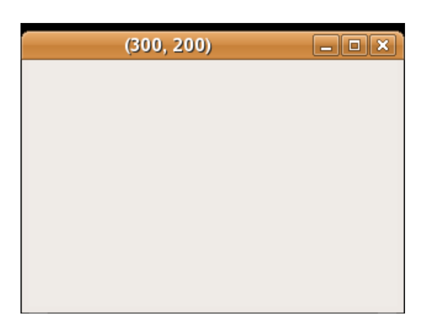


Figura: sizeevent.py

Evento Paint

Un evento paint es generado cuando una ventana es redibujada. Esto pasa cuando redimensionamos una ventana o cuando la maximizamos. Un evento paint pueden ser generado programáticamente también. Por ejemplo, cuando llamamos al método SetLabel() cambia un componente wx.StaticText. Note que cuando minimizamos una ventana, no se genera un evento paint.

```
#!/usr/bin/python
# paintevent.py

import wx

class PaintEvent(wx.Frame):
    def __init__(self, parent, id, title):
```

```
        wx.Frame.__init__(self, parent, id, title)

        self.count = 0
        self.Bind(wx.EVT_PAINT, self.OnPaint)
        self.Centre()
        self.Show(True)

    def OnPaint(self, event):
        self.count = self.count + 1
        print self.count


app = wx.App()
PaintEvent(None, -1, 'paintevent.py')
app.MainLoop()
```

En nuestro ejemplo imprimimos el número de eventos paint generados dentro de la consola.

Evento Key

Cuando pulsamos una tecla sobre nuestro teclado, wx.KeyEvent es generado. Este evento es enviado al componente que tiene el foco actualmente. Estos son los tres gestionadores asincrónicos diferentes de key:

- wx.EVT_KEY_DOWN
- wx.EVT_KEY_UP
- wx.EVT_CHAR

Un requerimietno común es para cerrar aplicaciones, cuando la tecla Esc es pulsada.

```
#!/usr/bin/python
# keyevent.py

import wx

class KeyEvent(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title)

        panel = wx.Panel(self, -1)
        panel.Bind(wx.EVT_KEY_DOWN, self.OnKeyDown)
        panel.SetFocus()

        self.Centre()
        self.Show(True)


    def OnKeyDown(self, event):
        keycode = event.GetKeyCode()
        if keycode == wx.WXK_ESCAPE:
            ret  = wx.MessageBox('Are you sure to quit?', 'Question',
                wx.YES_NO | wx.NO_DEFAULT, self)
            if ret == wx.YES:
                self.Close()
        event.Skip()
```

```
app = wx.App()
KeyEvent(None, -1, 'keyevent.py')
app.MainLoop()
```

```
 keycode = event.GetKeyCode()
```

Aquí tomamos el código de la tecla pulsada.

```
 if keycode == wx.WXK_ESCAPE:
```

Comprobamos el código de la tecla. La tecla Esc tiene el código *wx.WXK_ESCAPE*.

En este capítulo, hablamos acerca de eventos en wxPython.

Diálogos wxPython

Las ventanas de diálogo o diálogos son una parte indispensable de las más modernas GUI de aplicaciones. Un diálogo es definido como una conversación entre dos o más personas. En una aplicación de computadora un diálogo es una ventana la cual es usado para "hablar" a las aplicaciones. Un diálogo es usado para entrar datos, modificar datos, cambia la configuración de las aplicaciones etc. Los diálogos son medios importantes de comunicación entre a usuario y un programa de computadora.

Un caja de mensajes simple

Una caja de mensajes provee información reducida al usuario. Un buen ejemplo es una aplicación quemador de CD. Cuando un cd es terminado de quemar, una caja de mensajes aparece.

```
#!/usr/bin/python
# mensaje.py

import wx

class MessageDialog(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title)

        wx.FutureCall(5000, self.ShowMessage)

        self.Centre()
        self.Show(True)

    def ShowMessage(self):
        wx.MessageBox('Download completed', 'Info')


app = wx.App()
MessageDialog(None, -1, 'MessageDialog')
app.MainLoop()
```

```
wx.FutureCall(5000, self.ShowMessage)
```

wx.FutureCall llama un método después de 5 segundos. El primer parámetro es un valor de tiempo, Después del cual un método dado es llamado. El parámetro es en milisegundos. El segundo parámetro es un método a ser llamado.

```
def ShowMessage(self):
    wx.MessageBox('Download completed', 'Info')
```

wx.MessageBox muestra una ventana de diálogo pequeña. Proveemos tres parámetros. El texto del mensaje, el título del mensaje y finalmente el botón.

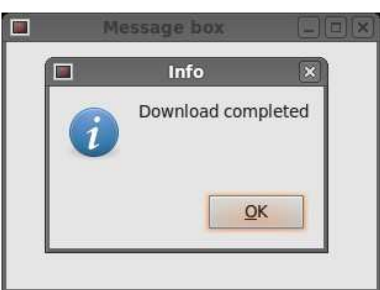


Figura: Un diálogo de mensaje

Diálogos predefinidos

wxPython tiene varios diálogos predefinidos. Estos son diálogos para tareas comunes de programación tales como mostrar texto, recibir entrada, cargar y guardar archivos etc.

Diálogos de mensajes

Diálogos de mensajes son usadas para mostrar mensajes al usuario. Aquellos son más flexibles que cuadros de mensajes simples, que vimos en el ejemplo previo. Aquellos son personabilizables. Podemos cambiar iconos y botones que se mostrarán en un Cuadro de Diálogo.

```
  wx.MessageDialog(wx.Window parent, string mensaje, string
caption=wx.MessageBoxCaptionStr, long style=wx.OK | wx.CANCEL | wx.CENTRE,
wx.Point pos=(-1, -1))
```

flag	meaning
wx.OK	muestra el botónOk
wx.CANCEL	muestra el botónCancel
wx.YES_NO	muestra los botones Sí, No
wx.YES_DEFAULT	Hace el botón Yes el por omisión
wx.NO_DEFAULT	Hace el botón No el por omisión
wx.ICON_EXCLAMATION	muestra un icono de alerta
wx.ICON_ERROR	muestra un icono de error
wx.ICON_HAND	Lo mismo que wx.ICON_ERROR
wx.ICON_INFORMATION	muestra un icono de información
wx.ICON_QUESTION	muestra un icono de pregunta

```
#!/usr/bin/python
# mensajes.py

import wx

class Messages(wx.Frame):
```

```
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(250, 150))

        panel = wx.Panel(self, -1)

        hbox = wx.BoxSizer()
        sizer = wx.GridSizer(2, 2, 2, 2)

        btn1 = wx.button(panel, -1, 'Info')
        btn2 = wx.button(panel, -1, 'Error')
        btn3 = wx.button(panel, -1, 'Question')
        btn4 = wx.button(panel, -1, 'Alert')

        sizer.AddMany([btn1, btn2, btn3, btn4])

        hbox.Add(sizer, 0, wx.ALL, 15)
        panel.SetSizer(hbox)


        btn1.Bind(wx.EVT_BUTTON, self.ShowMessage1)
        btn2.Bind(wx.EVT_BUTTON, self.ShowMessage2)
        btn3.Bind(wx.EVT_BUTTON, self.ShowMessage3)
        btn4.Bind(wx.EVT_BUTTON, self.ShowMessage4)

        self.Centre()
        self.Show(True)

    def ShowMessage1(self, event):
        dial = wx.MessageDialog(None, 'Download completed', 'Info', wx.OK)
        dial.ShowModal()

    def ShowMessage2(self, event):
        dial = wx.MessageDialog(None, 'Error loading file', 'Error', wx.OK |
            wx.ICON_ERROR)
        dial.ShowModal()

    def ShowMessage3(self, event):
        dial = wx.MessageDialog(None, 'Are you sure to quit?', 'Question',
            wx.YES_NO | wx.NO_DEFAULT | wx.ICON_QUESTION)
        dial.ShowModal()

    def ShowMessage4(self, event):
        dial = wx.MessageDialog(None, 'Unallowed operation', 'Exclamation',
wx.OK |
            wx.ICON_EXCLAMATION)
        dial.ShowModal()

app = wx.App()
Messages(None, -1, 'Messages')
app.MainLoop()
```

En nuestro ejemplo, tenemos creados cuatro botones y los ponemos en una grilla dimensionadora. Estos botones mostrarán cuatro diferentes ventanas de diálogo. Creamos ellos especificando diferentes banderas de estilo.

```
dial = wx.MessageDialog(None, 'Error loading file', 'Error', wx.OK |
```

```
    wx.ICON_ERROR)
dial.ShowModal()
```

La creación del diálogo de mensaje es simple. Fijamos el diálogo para ser una ventana de nivel tope proveyendo None como uno de los padres. El dos cadenas proveen el texto del mensaje y el título del diálogo. Mostraremos un botón OK y un icono de error especificando las banderas *wx.OK* y *wx.ICON_ERROR*. Para mostrar el diálogo sobre la pantalla, llamamos el método *ShowModal()*.



Diálogo de Información

Cuadro de Diálogo Acerca de

La mayoría de las aplicaciones tiene un típico cuadro de diálogo Acerca de. Éste es usualmente colocado en el menú de Ayuda. El propósito de este diálogo es para dar al usuario el información basica acerca del nombre y la versión de las aplicaciones. En el pasado, estos diálogos eran usados para ser muy breves. En estos días la mayoría de estos cuadros proveen información adicional acerca de los autores. Aquellos dan créditos para programadores adicionales o escritores de documentación. Aquellos también proveen información acerca de la licencia de las aplicaciones. Estos cuadros pueden mostrar el logo de la compañía o de las aplicaciones. Algunos de los más capaces cuadros Acerca de muestran animaciones. wxPython tiene un cuadro de diálogo especial Acerca de empezando con las series 2.8.x.

El cuadro de diálogo está localizada en el módulo Misc. En orden para crear un cuadro de diálogo Acerca de debemos crear dos objetos. Un *wx.AboutDialogInfo* y un *wx.AboutBox*.

```
wx.AboutDialogInfo()
```

Llamaremos los métodos siguientes sobre un objeto *wx.AboutDialogInfo* en nuestro ejemplo. Estos métodos son auto explicativos.

Método	Descripción
SetName(string name)	fija el nombre del programa
SetVersion(string version)	fija la versión del programa
SetDescription(string desc)	fija la descripción del programa
SetCopyright(string copyright)	fija el copyright del program
SetLicence(string licence)	fija la licencia del programa
SetIcon(wx.Icon icon)	fija el icono a ser mostrado
SetWebSite(string URL)	fija el sitio Web del programa
AddDeveloper(string developer)	Agrega un desarrollador a la lista de desarrolladores
AddDocWriter(string docwirter)	Agrega un documentador a la lista de escritores de documentación
AddArtist(string artist)	Agrega un artista a la lista de artistas
AddTranslator(string developer)	Agrega un desarrollador a la lista de traductores

El constructor del *wx.AboutBox* es el siguiente. Éste toma un *wx.AboutDialogInfo* como un parámetro.

```
wx.AboutBox(wx.AboutDialogInfo info)
```

wxPython pueden mostrar dos tipos de cuadros Acerca de. Esto depende de sobre cual plataforma estamos y cuales métodos llamamos. Esto puede ser un diálogo nativo o un Cuadro de Diálogo wxPython genérico. La ventana nativa del cuadro de diálogo Acerca de no puede mostrar iconos personalizados, texto de la licencia o las url's. Si omitimos estos tres campos, wxPython mostrarán un Cuadro de Diálogo nativo. De otra manera recurrirá a generar uno. Es imprudente proveer información de la licencia en un item de menú separado, si buscamos mantener tan nativa como sea posible. GTK+ pueden mostrar todos estos campos.

```
#!/usr/bin/python
# cuadro acerca de.py

import wx

ID_ABOUT = 1

class AboutDialogBox(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(260, 200))

        menubar = wx.MenuBar()
        help = wx.Menu()
        help.Append(ID_ABOUT, '&About')
        self.Bind(wx.EVT_MENU, self.OnAboutBox, id=ID_ABOUT)
        menubar.Append(help, '&Help')
        self.SetMenuBar(menubar)

        self.Centre()
        self.Show(True)

    def OnAboutBox(self, event):
        description = """File Hunter is an advanced file manager for the
Unix operating
system. Features include powerful built-in editor, advanced search
capabilities,
powerful batch renaming, file comparison, extensive archive handling y more.
"""

        licence = """File Hunter es free software; you can redistribute it
and/or modify it
under el terms del GNU General Public License as published por el Free
Software Foundation;
either version 2 del License, o (at your option) any después version.

File Hunter es distributed en el hope que it will be useful, pero WITHOUT
cualquier WARRANTY;
without even el implied warranty de MERCHANTABILITY o FITNESS Para un
PARTICULAR PURPOSE.
See el GNU General Public License for more detalles. You should tener
received a copy de
El GNU General Public License along con File Hunter; if not, write to
El Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
02111-1307  USA"""
```

```
        info = wx.AboutDialogInfo()

        info.SetIcon(wx.Icon('icons/hunter.png', wx.BITMAP_TYPE_PNG))
        info.SetName('File Hunter')
        info.SetVersion('1.0')
        info.SetDescription(description)
        info.SetCopyright('(C) 2007 jan bodnar')
        info.SetWebSite('http://www.zetcode.com')
        info.SetLicence(licence)
        info.AddDeveloper('jan bodnar')
        info.AddDocWriter('jan bodnar')
        info.AddArtist('The Tango crew')
        info.AddTranslator('jan bodnar')

        wx.AboutBox(info)


app = wx.App()
AboutDialogBox(None, -1, 'Cuadro de Diálogo Acerca de')
app.MainLoop()
```

```
        description = """File Hunter is an advanced file manager for the
Unix operating
system. Features include powerful built-in editor, advanced search
capabilities,
powerful batch renaming, file comparison, extensive archive handling y more.
"""
```

Éste no es la mejor idea para poner demasiado texto dentro del código de las aplicaciones. No quiero hacer el ejemplo demasiado complejo, por lo que pongo todo el texto dentro del código. Pero en programas de la vida real, el texto debería ser colocado separadamente en el interior de un archivo. Esto nos ayuda con el mantenimiento de nuestras aplicaciones. Por ejemplo, si buscamos traducir nuestras aplicaciones a otros languages.

```
info = wx.AboutDialogInfo()
```

La primer cosa para hacer es crear un objeto *wx.AboutDialogInfo*. El constructor esta vacio. Este no toma cualquier parámetro.

```
info.SetIcon(wx.Icon('icons/hunter.png', wx.BITMAP_TYPE_PNG))
info.SetName('File Hunter')
info.SetVersion('1.0')
info.SetDescription(description)
info.SetCopyright('(C) 2007 jan bodnar')
info.SetWebSite('http://www.zetcode.com')
info.SetLicence(licence)
info.AddDeveloper('jan bodnar')
info.AddDocWriter('jan bodnar')
info.AddArtist('El Tango crew')
info.AddTranslator('jan bodnar')
```

La siguiente cosa para hacer es llamar todos los métodos necesarios sobre el objeto *wx.AboutDialogInfo* creado.

```
wx.AboutBox(info)
```

Al final creamos un componente *wx.AboutBox*. El parámetro solamente que se necesita es el objeto *wx.AboutDialogInfo*.

Y por supuesto, si buscamos tener una animación o algunos otros elementos vistosos, debemos implementar nuestras diálogos Acerca de manualmente.



Cuadro de Diálogo Acerca de

Un diálogo personalizado

En el ejemplo siguiente creamos un Cuadro de Diálogo personalizado. Una aplicación para edición de imagenes puede cambiar la profundidad del color de una imagen. Para proveer esta funcionalidad, podríamos crear un Cuadro de Diálogo adecuado.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

'''
ZetCode wxPython tutorial

In this code example, we create a
custom dialog.

author: Jan Bodnar
website: www.zetcode.com
last modified: October 2011
'''

import wx

class ChangeDepthDialog(wx.Dialog):

    def __init__(self, parent, title):
        super(ChangeDepthDialog, self).__init__(parent=parent,
            title=title, size=(250, 200))

        panel = wx.Panel(self)
        vbox = wx.BoxSizer(wx.VERTICAL)

        sb = wx.StaticBox(panel, label='Colors')
        sbs = wx.StaticBoxSizer(sb, orient=wx.VERTICAL)
        sbs.Add(wx.RadioButton(panel, label='256 Colors',
```

```
style=wx.RB_GROUP))
        sbs.Add(wx.RadioButton(panel, label='16 Colors'))
        sbs.Add(wx.RadioButton(panel, label='2 Colors'))

        hbox1 = wx.BoxSizer(wx.HORIZONTAL)
        hbox1.Add(wx.RadioButton(panel, label='Custom'))
        hbox1.Add(wx.TextCtrl(panel), flag=wx.LEFT, border=5)
        sbs.Add(hbox1)

        panel.SetSizer(sbs)

        hbox2 = wx.BoxSizer(wx.HORIZONTAL)
        okButton = wx.Button(self, label='Ok')
        closeButton = wx.Button(self, label='Close')
        hbox2.Add(okButton)
        hbox2.Add(closeButton, flag=wx.LEFT, border=5)

        vbox.Add(panel, proportion=1, flag=wx.ALL|wx.EXPAND, border=5)
        vbox.Add(hbox2, flag= wx.ALIGN_CENTER|wx.TOP|wx.BOTTOM, border=10)

        self.SetSizer(vbox)

        okButton.Bind(wx.EVT_BUTTON, self.OnClose)
        closeButton.Bind(wx.EVT_BUTTON, self.OnClose)

    def OnClose(self, e):

        self.Destroy()


class Example(wx.Frame):

    def __init__(self, *args, **kwargs):
        super(Example, self).__init__(*args, **kwargs)

        self.InitUI()

    def InitUI(self):

        ID_DEPTH = wx.NewId()

        toolbar = self.CreateToolBar()
        toolbar.AddLabelTool(id=ID_DEPTH, label='',
            bitmap=wx.Bitmap('color.png'))

        self.Bind(wx.EVT_TOOL, self.OnChangeDepth, id=ID_DEPTH)

        self.SetSize((300, 200))
        self.SetTitle('Custom dialog')
        self.Centre()
        self.Show(True)

    def OnChangeDepth(self, e):

        chgdep = ChangeDepthDialog(None, title='Change Color Depth')
        chgdep.ShowModal()
```

```
        chgdep.Destroy()


def main():

    ex = wx.App()
    Example(None)
    ex.MainLoop()


if __name__ == '__main__':
    main()
```

En nuestro código de arriba, creamos un cuadro adaptado.

```
class ChangeDepth(wx.Dialog):
     def __init__(self, parent, id, title):
         wx.Dialog.__init__(self, parent, id, title, size=(250, 210))
```

En nuestro código de ejemplo creamos un Cuadro de Diálogo para cambiar la profundidad personalizada. Heredamos desde un componente *wx.Dialog*.

```
chgdep = ChangeDepth(None, -1, 'Change Color Depth')
chgdep.ShowModal()
chgdep.Destroy()
```

Instanciamos una clase ChangeDepth. Entonces llamamos el Cuadro de Diálogo *ShowModal()*. No debemos olvidar destruir nuestro Cuadro de Diálogo. Note la diferencia visual entre el diálogo y la ventana de máximo nivel. El diálogo en la siguiente figura tiene que ser activado. No podemos trabajar con el ventana de nivel superior hasta que el diálogo es destruido. Ésta es una clara diferencia en la barra de títulos de la ventanas.

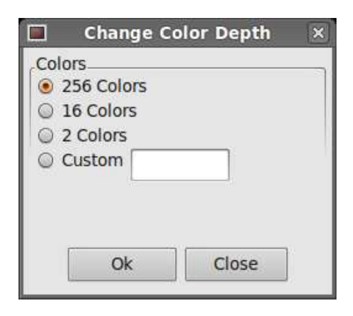


Figura: Un diálogo personalizado

En este capítulo, cubrimos diálogos.

Componentes

En esta sección, vamos a introducir componentes basicos en wxPython. Cada componente tendrá un pequeño código de ejemplo. Los componentes son bloques básicos de construcción de una aplicación. wxPythont tiene una amplia gama de componentes varios. Botones, casillas de verificación, deslizadores, cuadros de lista etc.

- wx.button

- wx.ToggleButton
- wx.BitmapButton
- wx.StaticLine
- wx.StaticText
- wx.StaticBox
- wx.ComboBox
- wx.CheckBox
- wx.StatusBar
- wx.RadioButton
- wx.Gauge
- wx.Slider
- wx.ListBox
- wx.SpinCtrl
- wx.SplitterWindow
- wx.ScrolledWindow
- wx.Notebook
- wx.Panel

wx.button

wx.button es un componente simple. Éste contiene una cadena de texto. Esto es usado para disparar una acción.

wx.button styles:

- wx.BU_LEFT
- wx.BU_TOP
- wx.BU_RIGHT
- wx.BU_BOTTOM
- wx.BU_EXACTFIT
- wx.NO_BORDER



Figura: botones.py

```
#!/usr/bin/python
# botones.py

import wx
import random

APP_SIZE_X = 300
APP_SIZE_Y = 200

class MyButtons(wx.Dialog):
    def __init__(self, parent, id, title):
```

```
        wx.Dialog.__init__(self, parent, id, title, size=(APP_SIZE_X,
APP_SIZE_Y))

        wx.button(self, 1, 'Close', (50, 130))
        wx.button(self, 2, 'Random Move', (150, 130), (110, -1))

        self.Bind(wx.EVT_BUTTON, self.OnClose, id=1)
        self.Bind(wx.EVT_BUTTON, self.OnRandomMove, id=2)

        self.Centre()
        self.ShowModal()
        self.Destroy()

    def OnClose(self, event):
        self.Close(True)

    def OnRandomMove(self, event):
        screensize = wx.GetDisplaySize()
        randx = random.randrange(0, screensize.x - APP_SIZE_X)
        randy = random.randrange(0, screensize.y - APP_SIZE_Y)
        self.Move((randx, randy))

app = wx.App(0)
MyButtons(None, -1, 'buttons.py')
app.MainLoop()
```

wx.ToggleButton

wx.ToggleButton es un botón que tiene dos estados. Presionado y no presionado. Usted alterna entre estos dos estados haciendo clic sobre él. Estas son situaciones donde esta funcionalidad se adapta bien.

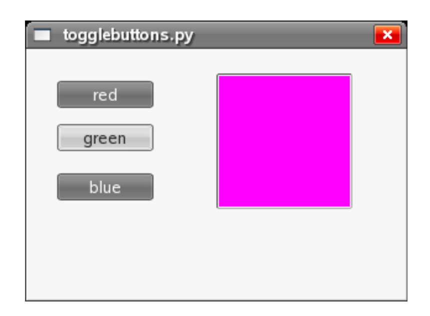


Figura: Togglebuttons.py

```
#!/usr/bin/python
# togglebuttons.py

import wx

class ToggleButtons(wx.Dialog):
    def __init__(self, parent, id, title):
        wx.Dialog.__init__(self, parent, id, title, size=(300, 200))

        self.colour = wx.Colour(0, 0, 0)

        wx.ToggleButton(self, 1, 'red', (20, 25))
        wx.ToggleButton(self, 2, 'green', (20, 60))
```

```
        wx.ToggleButton(self, 3, 'blue', (20, 100))

        self.panel  = wx.Panel(self, -1, (150, 20), (110, 110),
style=wx.SUNKEN_BORDER)
        self.panel.SetBackgroundColour(self.colour)

        self.Bind(wx.EVT_TOGGLEBUTTON, self.ToggleRed, id=1)
        self.Bind(wx.EVT_TOGGLEBUTTON, self.ToggleGreen, id=2)
        self.Bind(wx.EVT_TOGGLEBUTTON, self.ToggleBlue, id=3)

        self.Centre()
        self.ShowModal()
        self.Destroy()

    def ToggleRed(self, event):
        green = self.colour.Green()
        blue = self.colour.Blue()
        if  self.colour.Red():
            self.colour.Set(0, green, blue)
        else:
            self.colour.Set(255, green, blue)
        self.panel.SetBackgroundColour(self.colour)

    def ToggleGreen(self, event):
        red = self.colour.Red()
        blue = self.colour.Blue()
        if  self.colour.Green():
            self.colour.Set(red, 0, blue)
        else:
            self.colour.Set(red, 255, blue)
        self.panel.SetBackgroundColour(self.colour)

    def ToggleBlue(self, event):
        red = self.colour.Red()
        green = self.colour.Green()
        if  self.colour.Blue():
            self.colour.Set(red, green, 0)
        else:
            self.colour.Set(red, green, 255)
        self.panel.SetBackgroundColour(self.colour)


app = wx.App()
ToggleButtons(None, -1, 'togglebuttons.py')
app.MainLoop()
```

wx.BitmapButton

Un botón bitmap es un botón, que muestra un bitmap. Un botón bitmap pueden tener otros tres estados. Seleccionado, con foco y mostrado. Podemos fijar un bitmap específico para estos estados. Un reproductor de video es un buen ejemplo, donde los botones bitmap son usadas. Podemos ver aquí botones bitmap para reproducir, pausar, siguiente, previo y volúmen. Así creamos un esqueleto de a video reproductor en nuestro ejemplo siguiente.



Figura: Player.py

```
#!/usr/bin/python
# reproductor.py

import wx

class Player(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(350, 300))
        panel = wx.Panel(self, -1)

        pnl1 = wx.Panel(self, -1)
        pnl1.SetBackgroundColour(wx.BLACK)
        pnl2 = wx.Panel(self, -1 )

        menubar = wx.MenuBar()
        file = wx.Menu()
        play = wx.Menu()
        view = wx.Menu()
        tools = wx.Menu()
        favorites = wx.Menu()
        help = wx.Menu()

        file.Append(101, '&quit', 'Quit aplications')

        menubar.Append(file, '&File')
        menubar.Append(play, '&Play')
        menubar.Append(view, '&View')
        menubar.Append(tools, '&Tools')
        menubar.Append(favorites, 'F&avorites')
        menubar.Append(help, '&Help')

        self.SetMenuBar(menubar)

        slider1 = wx.Slider(pnl2, -1, 0, 0, 1000)
        pausar = wx.BitmapButton(pnl2, -1, wx.Bitmap('icons/stock_media-
pause.png'))
        play  = wx.BitmapButton(pnl2, -1, wx.Bitmap('icons/stock_media-
play.png'))
        siguiente  = wx.BitmapButton(pnl2, -1, wx.Bitmap('icons/stock_media-
```

```
next.png'))
        prev  = wx.BitmapButton(pnl2, -1, wx.Bitmap('icons/stock_media-
prev.png'))
        volúmen = wx.BitmapButton(pnl2, -1, wx.Bitmap('icons/volume.png'))
        slider2 = wx.Slider(pnl2, -1, 0, 0, 100, size=(120, -1))

        vbox = wx.BoxSizer(wx.VERTICAL)
        hbox1 = wx.BoxSizer(wx.HORIZONTAL)
        hbox2 = wx.BoxSizer(wx.HORIZONTAL)

        hbox1.Add(slider1, 1)
        hbox2.Add(pause)
        hbox2.Add(play, flag=wx.RIGHT, border=5)
        hbox2.Add(next, flag=wx.LEFT, border=5)
        hbox2.Add(prev)
        hbox2.Add((-1, -1), 1)
        hbox2.Add(volume)
        hbox2.Add(slider2, flag=wx.TOP | wx.LEFT, border=5)

        vbox.Add(hbox1, flag=wx.EXPAND | wx.BOTTOM, border=10)
        vbox.Add(hbox2, 1, wx.EXPAND)
        pnl2.SetSizer(vbox)

        sizer = wx.BoxSizer(wx.VERTICAL)
        sizer.Add(pnl1, 1, flag=wx.EXPAND)
        sizer.Add(pnl2, flag=wx.EXPAND | wx.BOTTOM | wx.TOP, border=10)

        self.SetMinSize((350, 300))
        self.CreateStatusBar()
        self.SetSizer(sizer)

        self.Centre()
        self.Show()


app = wx.App()
Player(None, -1, 'Player')
app.MainLoop()
```

```
 pausar = wx.BitmapButton(pnl2, -1, wx.Bitmap('icons/stock_media-
pause.png'))
```

La creación del *wx.BitmapButton* es auto explicativa.

```
 hbox2.Add(prev)
 hbox2.Add((-1, -1), 1)
 hbox2.Add(volume)
```

Aquí ponemos algo de espacio entre el botón previo y el botón volúmen. La proporción es fijada a 1. De esta manera, el espacio crecerá, cuando redimensionemos la ventana.

```
self.SetMinSize((350, 300))
```

Aquí fijamos el tamaño mínimo del reproductor. No tiene mucho sentido reducir el tamaño la ventana debajo de algunos valores.

wx.StaticLine

Este componente muestra una línea simple sobre la ventana. Esta puede ser horizontal o vertical. El guión centraleurope.py muestra paises europeos centrales y su población. El wx.StatLine hace que parezca más visualmente atractivo.

wx.StaticLine styles

- wx.LI_HORIZONTAL
- wx.LI_VERTICAL



Figura: CentralEurope.py

```
#!/usr/bin/python
# centraleurope.py

import wx

class CentralEurope(wx.Dialog):
    def __init__ (self, parent, ID, title):
        wx.Dialog.__init__(self, parent, ID, title, size=(360, 370))

        font = wx.Font(10, wx.DEFAULT, wx.NORMAL, wx.BOLD)
        heading = wx.StaticText(self, -1, 'El Central Europe', (130, 15))
        heading.SetFont(font)

        wx.StaticLine(self, -1, (25, 50), (300,1))

        wx.StaticText(self, -1, 'Slovakia', (25, 80))
        wx.StaticText(self, -1, 'Hungary', (25, 100))
        wx.StaticText(self, -1, 'Poland', (25, 120))
        wx.StaticText(self, -1, 'Czech Republic', (25, 140))
        wx.StaticText(self, -1, 'Germany', (25, 160))
        wx.StaticText(self, -1, 'Slovenia', (25, 180))
        wx.StaticText(self, -1, 'Austria', (25, 200))
        wx.StaticText(self, -1, 'Switzerland', (25, 220))

        wx.StaticText(self, -1, '5 379 000', (250, 80))
        wx.StaticText(self, -1, '10 084 000', (250, 100))
        wx.StaticText(self, -1, '38 635 000', (250, 120))
        wx.StaticText(self, -1, '10 240 000', (250, 140))
```

```
        wx.StaticText(self, -1, '82 443 000', (250, 160))
        wx.StaticText(self, -1, '2 001 000', (250, 180))
        wx.StaticText(self, -1, '8 032 000', (250, 200))
        wx.StaticText(self, -1, '7 288 000', (250, 220))

        wx.StaticLine(self, -1, (25, 260), (300,1))

        sum = wx.StaticText(self, -1, '164 102 000', (240, 280))
        sum_font = sum.GetFont()
        sum_font.SetWeight(wx.BOLD)
        sum.SetFont(sum_font)

        wx.button(self, 1, 'Ok', (140, 310), (60, 30))

        self.Bind(wx.EVT_BUTTON, self.OnOk, id=1)

        self.Centre()
        self.ShowModal()
        self.Destroy()

    def OnOk(self, event):
        self.Close()

app = wx.App()
CentralEurope(None, -1, 'centraleurope.py')
app.MainLoop()
```

wx.StaticText

Un componente wx.StaticText muestra uno o más líneas de texto de solo lectura.

wx.StaticText Styles

- wx.ALIGN_RIGHT
- iwx.ALIGN_LEFT
- wx.ALIGN_CENTER / wx.ALIGN_CENTRE
- wx.ST_NO_AUTORESIZE

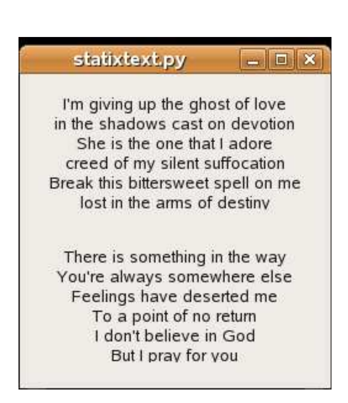


Figura: statictext.py

```
#!/usr/bin/python
# statictext.py

import wx
```

```
class StaticText(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title)

        lyrics1 = '''I'm giving up el ghost de love
in el shadows cast on devotion
She es el uno que I adore
creed de my silent suffocation
Break this bittersweet spell on me
lost en el arms de destiny'''

        lyrics2 = '''This is something en the way
You're always somewhere else
Feelings have deserted me
To a point de no return
I don't believe en God
But I pray for you'''

        vbox = wx.BoxSizer(wx.VERTICAL)
        panel = wx.Panel(self, -1)
        st1 = wx.StaticText(panel, -1, lyrics1, style=wx.ALIGN_CENTRE)
        st2 = wx.StaticText(panel, -1, lyrics2, style=wx.ALIGN_CENTRE)
        vbox.Add(st1, 1, wx.EXPAND |  wx.TOP | wx.BOTTOM, 15)
        vbox.Add(st2, 1, wx.EXPAND |  wx.TOP | wx.BOTTOM, 15)
        panel.SetSizer(vbox)
        self.Centre()
        self.Show(True)


app = wx.App()
StaticText(None, -1, 'statixtext.py')
app.MainLoop()
```

wx.StaticBox

Éste es un tipo de un componente decorador. Esto es usado para agrupar logicamente varios componentes. Note que este componente debe ser creado antes que los componentes que lo contiene, y que estos componentes deberían ser hermanos, no hijos, de la caja estática.



Figura: staticbox.py

```
#!/usr/bin/python
# staticbox.py

import wx
```

```
class StaticBox(wx.Dialog):
    def __init__(self, parent, id, title):
        wx.Dialog.__init__(self, parent, id, title, size=(250, 230))

        wx.StaticBox(self, -1, 'Personal Info', (5, 5), size=(240, 170))
        wx.CheckBox(self, -1 ,'Male', (15, 30))
        wx.CheckBox(self, -1 ,'Married', (15, 55))
        wx.StaticText(self, -1, 'Age', (15, 95))
        wx.SpinCtrl(self, -1, '1', (55, 90), (60, -1), min=1, max=120)
        wx.button(self, 1, 'Ok', (90, 185), (60, -1))

        self.Bind(wx.EVT_BUTTON, self.OnClose, id=1)

        self.Centre()
        self.ShowModal()
        self.Destroy()

    def OnClose(self, event):
        self.Close()

app = wx.App()
StaticBox(None, -1, 'staticbox.py')
app.MainLoop()
```

wx.ComboBox

wx.ComboBox es una combinación de un campo de texto en una simple línea, un botón con una imagen de flecha abajo y una caja de lista. Cuando usted pulsa el botón, aparece una caja de lista. El usuario puede seleccionar solamente una opción desde la lista de cadenas suministrada.

wx.ComboBox tiene el siguiente constructor:

```
wx.ComboBox(int id, string value='', wx.Point pos=wx.DefaultPosition,
wx.Size size=wx.DefaultSize,
  wx.List choices=wx.EmptyList, int style=0, wx.Validator
validator=wx.DefaultValidator,
  string name=wx.ComboBoxNameStr)
```

wx.ComboBox styles

- wx.CB_DROPDOWN
- wx.CB_READONLY
- wx.CB_SORT



Figura: combobox.py

```
#!/usr/bin/python
# combobox.py

import wx

class ComboBox(wx.Dialog):
    def __init__(self, parent, id, title):
        wx.Dialog.__init__(self, parent, id, title, size=(250, 270))

        panel = wx.Panel(self, -1, (75, 20), (100, 127),
style=wx.SUNKEN_BORDER)
        self.picture = wx.StaticBitmap(panel)
        panel.SetBackgroundColour(wx.WHITE)

        self.images = ['tolstoy.jpg', 'feuchtwanger.jpg', 'balzac.jpg',
'pasternak.jpg',
                    'galsworthy.jpg', 'wolfe.jpg', 'zweig.jpg']
        authors = ['Leo Tolstoy', 'Lion Feuchtwanger', 'Honore de Balzac',
                'Boris Pasternak', 'John Galsworthy', 'Tom Wolfe', 'Stefan
Zweig']

        wx.ComboBox(self, -1, pos=(50, 170), size=(150, -1),
choices=authors,
                                style=wx.CB_READONLY)
        wx.button(self, 1, 'Close', (80, 220))

        self.Bind(wx.EVT_BUTTON, self.OnClose, id=1)
        self.Bind(wx.EVT_COMBOBOX, self.OnSelect)

        self.Centre()
        self.ShowModal()
        self.Destroy()

    def OnClose(self, event):
        self.Close()

    def OnSelect(self, event):
        item = event.GetSelection()
        self.picture.SetFocus()
        self.picture.SetBitmap(wx.Bitmap('images/' + self.images[item]))


app = wx.App()
```

```
ComboBox(None, -1, 'combobox.py')
app.MainLoop()
```

wx.CheckBox

wx.CheckBox es un componente que tiene dos estados. Prendido y Apagado. Esto es una caja con una etiqueta. La etiqueta pueden ser fijada a la derecha o a la izquierda de la caja. Si la casilla de verificación es tildada, éste es representado por un tilde en una caja.

wx.CheckBox Styles

- wx.ALIGN_RIGHT



Figura: checkbox.py

```
#!/usr/bin/python
# checkbox.py

import wx

class CheckBox(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(250, 170))

        panel = wx.Panel(self, -1)
        self.cb = wx.CheckBox(panel, -1, 'Show Title', (10, 10))
        self.cb.SetValue(True)

        wx.EVT_CHECKBOX(self, self.cb.GetId(), self.ShowTitle)

        self.Show()
        self.Centre()

    def ShowTitle(self, event):
        if self.cb.GetValue():
            self.SetTitle('checkbox.py')
        else: self.SetTitle('')


app = wx.App()
CheckBox(None, -1, 'checkbox.py')
app.MainLoop()
```

wx.StatusBar

Como su nombre lo indica, el componente wx.StatusBar es usado para mostrar la información del estado de las aplicaciones. Esto puede ser dividido dentro de varias partes para mostrar diferentes tipos

de información. Podemos insertar otros componentes dentro del wx.StatusBar. Esto puede ser usado como una alternativa a los diálogos, dado que diálogos pueden ser abusivos y molestos para la mayoría de los usuarios. Podemos crear un wx.StatusBar por dos caminos. Podemos manualmente crear nuestras propias wx.StatusBar y llamar al método SetStatusBar() o podemos simplemente llamar al método CreateStatusBar(). El último método crea un wx.StatusBar por omisión para nosotros. En nuestro ejemplo, tenemos un componente wx.Frame y otros cinco componentes. Si pasamos un puntero del ratón sobre un componente, su descripción es mostrada sobre el wx.StatusBar.

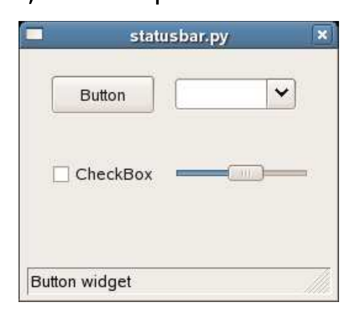


Figura: statusbar.py

```
#!/usr/bin/python
# statusbar.py

import wx

class Statusbar(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(250, 200),
        style=wx.CAPTION | wx.SYSTEM_MENU | wx.CLOSE_BOX)

        panel = wx.Panel(self, 1)

        button = wx.button(panel, 2, 'botón', (20, 20))
        text = wx.CheckBox(panel, 3, 'CheckBox', (20, 90))
        combo = wx.ComboBox(panel, 4, '', (120, 22))
        slider = wx.Slider(panel, 5, 6, 1, 10, (120, 90), (110, -1))

        panel.Bind(wx.EVT_ENTER_WINDOW, self.EnterPanel, id=1)
           button.Bind(wx.EVT_ENTER_WINDOW, self.EnterButton, id=2)
        text.Bind(wx.EVT_ENTER_WINDOW, self.EnterText, id=3)
        combo.Bind(wx.EVT_ENTER_WINDOW, self.EnterCombo, id=4)
        slider.Bind(wx.EVT_ENTER_WINDOW, self.EnterSlider, id=5)

        self.sb = self.CreateStatusBar()
        self.SetMaxSize((250, 200))
        self.SetMinSize((250, 200))
        self.Show(True)
        self.Centre()

    def EnterButton(self, event):
        self.sb.SetStatusText('botón widget')
        event.Skip()

    def EnterPanel(self, event):
        self.sb.SetStatusText('Panel widget')
        event.Skip()
```

```
    def EnterText(self, event):
        self.sb.SetStatusText('CheckBox widget')
        event.Skip()

    def EnterCombo(self, event):
        self.sb.SetStatusText('ComboBox widget')
        event.Skip()

    def EnterSlider(self, event):
        self.sb.SetStatusText('Slider widget')
        event.Skip()

app = wx.App()
Statusbar(None, -1, 'statusbar.py')
app.MainLoop()
```

wx.RadioButton

wx.RadioButton es un componente que permite el usuario seleccionar una simple elección exclusiva desde un grupo de opciones. Un grupo de botones de radio está definido por tener el primer RadioButton en el grupo que contienen el estilo wx.RB_GROUP. Todo los otros RadioButtons definidos después del primer RadioButton con esta bandera de estilo podrán ser agregados al grupo funcional del primer RadioButton. Declarando otro RadioButton con la bandera wx.RB_GROUP se iniciarán un nuevo conjunto de botones de radio.

wx.RadioButton Styles

- wx.RB_GROUP
- wx.RB_SINGLE
- wx.CB_SORT



Figura: radiobutton.py

```
#!/usr/bin/python
# radiobuttons.py

import wx

class RadioButtons(wx.Frame):
    def __init__(self, parent, id, title, size=(210, 150)):
        wx.Frame.__init__(self, parent, id, title)
        panel = wx.Panel(self, -1)
        self.rb1 = wx.RadioButton(panel, -1, 'Value A', (10, 10),
style=wx.RB_GROUP)
        self.rb2 = wx.RadioButton(panel, -1, 'Value B', (10, 30))
        self.rb3 = wx.RadioButton(panel, -1, 'Value C', (10, 50))

        self.Bind(wx.EVT_RADIOBUTTON, self.SetVal, id=self.rb1.GetId())
```

```
        self.Bind(wx.EVT_RADIOBUTTON, self.SetVal, id=self.rb2.GetId())
        self.Bind(wx.EVT_RADIOBUTTON, self.SetVal, id=self.rb3.GetId())

        self.statusbar = self.CreateStatusBar(3)
        self.SetVal(True)
        self.Centre()
        self.Show(True)

    def SetVal(self, event):
        state1 = str(self.rb1.GetValue())
        state2 = str(self.rb2.GetValue())
        state3 = str(self.rb3.GetValue())

        self.statusbar.SetStatusText(state1, 0)
        self.statusbar.SetStatusText(state2, 1)
        self.statusbar.SetStatusText(state3, 2)


app = wx.App()
RadioButtons(None, -1, 'radiobuttons.py')
app.MainLoop()
```

wx.Gauge

wx.Gauge es un componente que es usado, cuando procesamos tareas largas.

wx.Gauge styles

- wx.GA_HORIZONTAL
- wx.GA_VERTICAL



Figura: gauge.py

```
# gauge.py

import wx

class Gauge(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(300, 200))

        self.timer = wx.Timer(self, 1)
        self.count = 0
```

```
        self.Bind(wx.EVT_TIMER, self.OnTimer, self.timer)

        panel = wx.Panel(self, -1)
        vbox = wx.BoxSizer(wx.VERTICAL)
        hbox1 = wx.BoxSizer(wx.HORIZONTAL)
        hbox2 = wx.BoxSizer(wx.HORIZONTAL)
        hbox3 = wx.BoxSizer(wx.HORIZONTAL)

        self.gauge = wx.Gauge(panel, -1, 50, size=(250, 25))
        self.btn1 = wx.button(panel, wx.ID_OK)
        self.btn2 = wx.button(panel, wx.ID_STOP)
        self.text = wx.StaticText(panel, -1, 'Task para ser done')

        self.Bind(wx.EVT_BUTTON, self.OnOk, self.btn1)
        self.Bind(wx.EVT_BUTTON, self.OnStop, self.btn2)

        hbox1.Add(self.gauge, 1, wx.ALIGN_CENTRE)
        hbox2.Add(self.btn1, 1, wx.RIGHT, 10)
        hbox2.Add(self.btn2, 1)
        hbox3.Add(self.text, 1)
        vbox.Add((0, 30), 0)
        vbox.Add(hbox1, 0, wx.ALIGN_CENTRE)
        vbox.Add((0, 20), 0)
        vbox.Add(hbox2, 1, wx.ALIGN_CENTRE)
        vbox.Add(hbox3, 1, wx.ALIGN_CENTRE)

        panel.SetSizer(vbox)
        self.Centre()
        self.Show(True)

    def OnOk(self, event):
        if self.count >= 50:
            return
        self.timer.Start(100)
        self.text.SetLabel('Task en Progress')

    def OnStop(self, event):
        if self.count == 0 o self.count >= 50 o not self.timer.IsRunning():
            return
        self.timer.Stop()
        self.text.SetLabel('Task Interrupted')
        wx.Bell()

    def OnTimer(self, event):
        self.count = self.count +1
        self.gauge.SetValue(self.count)
        if self.count == 50:
            self.timer.Stop()
            self.text.SetLabel('Task Completed')

app = wx.App()
Gauge(None, -1, 'gauge.py')
app.MainLoop()
```

wx.Slider

wx.Slider es un componente que tiene a manejo simple. Este manejador pueden ser empujado hacia adelante y atras. De esta manera estamos eligiendo un valor para una tarea específica. Decimos que buscamos entrar dentro de nuestra aplicación la edad de un cliente. Para este propósito, wx.Slider podría ser una buena elección.

wx.Slider styles

- wxSL_HORIZONTAL
- wxSL_VERTICAL
- wxSL_AUTOTICKS
- wxSL_LABELS
- wxSL_LEFT
- wxSL_RIGHT
- wxSL_TOP
- wxSL_BOTTOM
- wxSL_INVERSE



Figura: slider.py

```
#!/usr/bin/python
# slider.py

import wx

class Slider(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(300, 150))

        panel = wx.Panel(self, -1)
        vbox = wx.BoxSizer(wx.VERTICAL)
        hbox = wx.BoxSizer(wx.HORIZONTAL)

        self.sld = wx.Slider(panel, -1, 200, 150, 500, (-1, -1), (250, -1),
                wx.SL_AUTOTICKS | wx.SL_HORIZONTAL | wx.SL_LABELS)
        btn1 = wx.button(panel, 1, 'Adjust')
        btn2 = wx.button(panel, 2,  'Close')

        wx.EVT_BUTTON(self, 1, self.OnOk)
        wx.EVT_BUTTON(self, 2, self.OnClose)

        vbox.Add(self.sld, 1, wx.ALIGN_CENTRE)
        hbox.Add(btn1, 1, wx.RIGHT, 10)
        hbox.Add(btn2, 1)
        vbox.Add(hbox, 0, wx.ALIGN_CENTRE | wx.ALL, 20)
        panel.SetSizer(vbox)
        self.Centre()
        self.Show(True)
```

```
    def OnOk(self, event):
        val = self.sld.GetValue()
        self.SetSize((val*2, val))

    def OnClose(self, event):
        self.Close()

app = wx.App()
Slider(None, -1, 'slider.py')
app.MainLoop()
```

wx.ListBox

wx.Listbox es un componente que consiste de una caja de deslizamiento y una lista de items. El usuario puede seleccionar uno o más items desde esta lista. Esto depende de sobre si éste es creado como una caja de selección simple o múltiple. Los items seleccionados son marcados.

El ejemplo listbox.py consiste de los cuatro componentes diferentes. wx.Listbox, wx.TextCtrl, wx.StaticText y wx.button. Los componentes son organizados con dimensionadores. wx.Listbox tiene una lista de seis diferentes tipos. Estas abreviaturas son explicadas en el segundo wx.TextCtrl. Esta vez es mostrada en el componente wx.StaticText. El componente wx.Timer es usado para actualizar el tiempo cada 100 milisegundos.

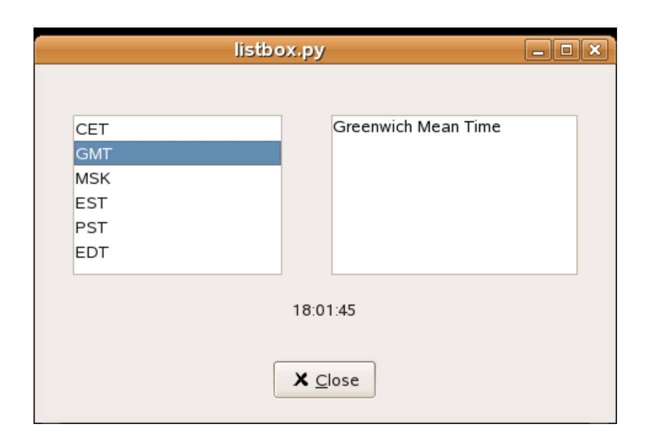


Figura: listbox.py

```
#!/usr/bin/python
# listbox.py

import wx
from time import *

class Listbox(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(550, 350))

        zone_list = ['CET', 'GMT', 'MSK', 'EST', 'PST', 'EDT']

        self.full_list = {
            'CET': 'Central European Time',
            'GMT': 'Greenwich Mean Time',
            'MSK': 'Moscow Time',
```

```
            'EST': 'Eastern standars Time',
            'PST': 'Pacific standars Time',
            'EDT': 'Eastern Daylight Time'
        }

        self.time_diff = {
            'CET' : 1,
            'GMT' : 0,
            'MSK': 3,
            'EST': -5,
            'PST': -8,
            'EDT': -4
        }

        vbox = wx.BoxSizer(wx.VERTICAL)
        hbox1 = wx.BoxSizer(wx.HORIZONTAL)
        hbox2 = wx.BoxSizer(wx.HORIZONTAL)
        hbox3 = wx.BoxSizer(wx.HORIZONTAL)

        self.timer = wx.Timer(self, 1)
        self.diff = 0

        panel = wx.Panel(self, -1)

        self.time_zones = wx.ListBox(panel, 26, (-1, -1), (170, 130),
                zone_list, wx.LB_SINGLE)
        self.time_zones.SetSelection(0)
        self.text = wx.TextCtrl(panel, -1, 'Central European Time',
                size=(200, 130), style=wx.TE_MULTILINE)
        self.time = wx.StaticText(panel, -1, '')
        btn = wx.button(panel, wx.ID_CLOSE, 'Close')
        hbox1.Add(self.time_zones, 0, wx.TOP, 40)
        hbox1.Add(self.text, 1, wx.LEFT | wx.TOP, 40)
        hbox2.Add(self.time, 1, wx.ALIGN_CENTRE)
        hbox3.Add(btn, 0, wx.ALIGN_CENTRE)
        vbox.Add(hbox1, 0, wx.ALIGN_CENTRE)
        vbox.Add(hbox2, 1, wx.ALIGN_CENTRE)
        vbox.Add(hbox3, 1, wx.ALIGN_CENTRE)

        panel.SetSizer(vbox)

        self.timer.Start(100)

        wx.EVT_BUTTON(self, wx.ID_CLOSE, self.OnClose)
        wx.EVT_LISTBOX(self, 26, self.OnSelect)
        wx.EVT_TIMER(self, 1, self.OnTimer)

        self.Show(True)
        self.Centre()

    def OnClose(self, event):
        self.Close()

    def OnSelect(self, event):
        index = event.GetSelection()
        time_zone = self.time_zones.GetString(index)
```

```
        self.diff = self.time_diff[time_zone]
        self.text.SetValue(self.full_list[time_zone])

    def OnTimer(self, event):
        ct = gmtime()
        print_time = (ct[0], ct[1], ct[2], ct[3]+self.diff,
                        ct[4], ct[5], ct[6], ct[7], -1)
        self.time.SetLabel(strftime("%H:%M:%S", print_time))


app = wx.App()
Listbox(None, -1, 'listbox.py')
app.MainLoop()
```

wx.SpinCtrl

Este componente le permite incrementar y decrementar un valor. Éste tiene dos botones flechas, una arriba y la otra abajo, para este propósito. El usuario puede entrar un valor dentro de una caja o incrementar/decrementar éste por estas dos flechas. El guión Converter convierte temperatura Fahrenheit a Celsius. Este ejemplo es muy popular y puede ser encontrado en la mayoría de los libros de programación iniciales. Así hice un ejemplo wxPython también.

wx.SpinCtrl styles

- wx.SP_ARROW_KEYS
- wx.SP_WRAP

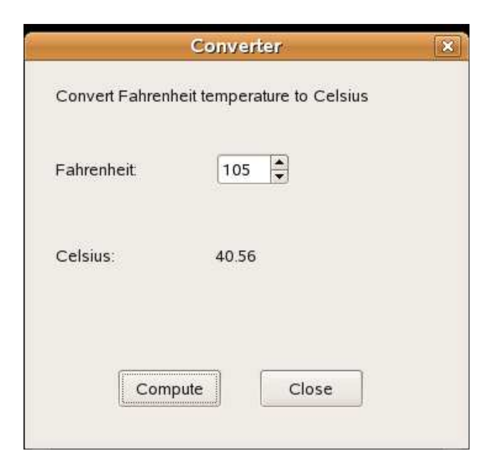


Figura: converter.py

```
#!/usr/bin/python
# spinctrl.py

import wx

class Converter(wx.Dialog):
    def __init__(self, parent, id, title):
        wx.Dialog.__init__(self, parent, id, title, size=(350, 310))

        wx.StaticText(self, -1, 'Convert Fahrenheit temperature to Celsius',
(20,20))
        wx.StaticText(self, -1, 'Fahrenheit: ', (20, 80))
        wx.StaticText(self, -1, 'Celsius: ', (20, 150))
        self.celsius =  wx.StaticText(self, -1, '', (150, 150))
```

```
        self.sc = wx.SpinCtrl(self, -1, '',  (150, 75), (60, -1))
        self.sc.SetRange(-459, 1000)
        self.sc.SetValue(0)
        compute_btn = wx.button(self, 1, 'Compute', (70, 250))
        compute_btn.SetFocus()
        clear_btn = wx.button(self, 2, 'Close', (185, 250))

        wx.EVT_BUTTON(self, 1, self.OnCompute)
        wx.EVT_BUTTON(self, 2, self.OnClose)
        wx.EVT_CLOSE(self, self.OnClose)

        self.Centre()
        self.ShowModal()
        self.Destroy()

    def OnCompute(self, event):
        fahr = self.sc.GetValue()
        cels = round((fahr-32)*5/9.0, 2)
        self.celsius.SetLabel(str(cels))

    def OnClose(self, event):
        self.Destroy()

app = wx.App()
Converter(None, -1, 'Converter')
app.MainLoop()
```

wx.SplitterWindow

Este componente permite dividir el área principal de una aplicación dentro de partes. El usuario puede dinámicamente redimensionar estos partes con el puntero del ratón. Como una solución pueden ser vistos en clientes de correo (evolution) o en software de quemado (k3b). Usted puede dividir un área verticalmente u horizontalmente.



Figura: splitterwindow.py

```
#!/usr/bin/python
# splitterwindow.py

import wx

class Splitterwindow(wx.Frame):
```

```
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(350, 300))

        quote = '''Whether you think que you can, o que you can't, you are 
usually right'''

        splitter = wx.SplitterWindow(self, -1)
        panel1 = wx.Panel(splitter, -1)
        wx.StaticText(panel1, -1, quote, (100, 100), style=wx.ALIGN_CENTRE)

        panel1.SetBackgroundColour(wx.LIGHT_GREY)
        panel2 = wx.Panel(splitter, -1)
        panel2.SetBackgroundColour(wx.WHITE)
        splitter.SplitVertically(panel1, panel2)
        self.Centre()
        self.Show(True)

app = wx.App()
Splitterwindow(None, -1, 'splitterwindow.py')
app.MainLoop()
```

wx.ScrolledWindow

Éste es uno de los componentes contenedores. Esto puede ser útil, cuando tenemos un área más grande que una ventana pueda mostrar. En nuestro ejemplo, demostramos este caso. Colocamos una gran imagen dentro de nuestra ventana. Cuando la ventana es menor que nuestra imagen, barras de desplazamiento son mostradas automáticamente.



Figura: scrolledwindow.py

```
#!/usr/bin/python
# scrolledwindow.py

import wx
```

```
class ScrolledWindow(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(500, 400))

        sw = wx.ScrolledWindow(self)
        bmp =
wx.Image('images/aliens.jpg',wx.BITMAP_TYPE_JPEG).ConvertToBitmap()
        wx.StaticBitmap(sw, -1, bmp)
        sw.SetScrollbars(20, 20, 55, 40)
        sw.Scroll(50,10)
        self.Centre()
        self.Show()

app = wx.App()
ScrolledWindow(None, -1, 'Aliens')
app.MainLoop()
```

El método SetScrollbars() crea barras de desplazmiento horizontal y vertical. Al llamar el método Scroll() nosotros por programación desplazamos la posición dada.

wx.Notebook

El componente wx.Notebook junta ventanas múltiples con las fichas correspondientes.

Usted puede posicionar el componente Notebook usando las siguientes banderas de estilo:

- wx.NB_LEFT
- wx.NB_RIGHT
- wx.NB_TOP
- wx.NB_BOTTOM

El por omisión position es wx.NB_TOP.

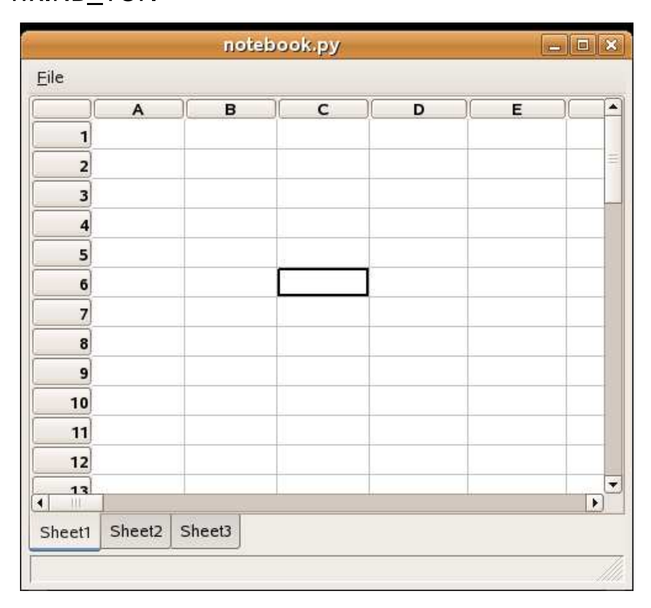


Figura: notebook.py

```
#!/usr/bin/python
# notebook.py
```

```
import wx
import wx.lib.sheet as sheet


class MySheet(sheet.CSheet):
    def __init__(self, parent):
        sheet.CSheet.__init__(self, parent)
        self.SetNumberRows(50)
        self.SetNumberCols(50)

class Notebook(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(600, 500))
        menubar = wx.MenuBar()
        file = wx.Menu()
        file.Append(101, 'Quit', '' )
        menubar.Append(file, '&File')
        self.SetMenuBar(menubar)

        wx.EVT_MENU(self, 101, self.OnQuit)

        nb = wx.Notebook(self, -1, style=wx.NB_BOTTOM)
        self.sheet1 = MySheet(nb)
        self.sheet2 = MySheet(nb)
        self.sheet3 = MySheet(nb)

        nb.AddPage(self.sheet1, 'Sheet1')
        nb.AddPage(self.sheet2, 'Sheet2')
        nb.AddPage(self.sheet3, 'Sheet3')

        self.sheet1.SetFocus()
        self.StatusBar()
        self.Centre()
        self.Show()

    def StatusBar(self):
        self.statusbar = self.CreateStatusBar()

    def OnQuit(self, event):
        self.Close()

app = wx.App()
Notebook(None, -1, 'notebook.py')
app.MainLoop()
```

En nuestro ejemplo creamos un componente Notebook con el estilo wx.NB_BOTTOM. Éste es por lo tanto posicionado sobre el fondo del Marcos en consecuencia. Agregaremos varios componentes dentro del componente notebook con el método AddPage(). Ponemos un simple componente hoja de cálculo. El componente Spreadsheet puede ser encontrado en el módulo wx.lib.sheet.

wx.Panel

wx.Panel es un componente padre basico. Éste agrega algunas funcionalidades básicas al componente *wx.Window*, el cual no son por lo general usados directamente. Normalmente creamos primero un componente wx.Frame. Entonces colocamos un componente *wx.Panel* en el interior de este marco.

Entonces colocamos componentes sobre el panel. Éste es el escenario común. Sin embargo, podemos también combinar paneles para crear interesante interfaz. En el siguientes ejemplo creamos a dos side ventana con cabeceras. Usamos altogether seis diferentes *wx.Panel* componentes.

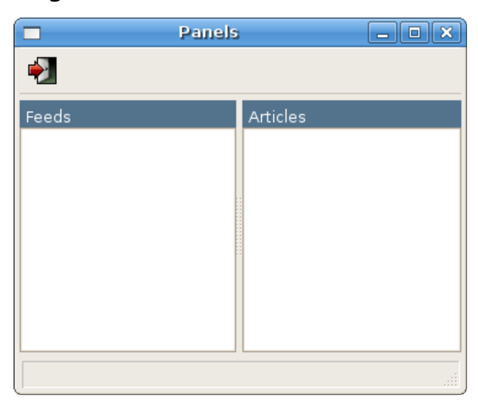


Figura: paneles.py

```
#!/usr/bin/python
# paneles.py

import wx


class Panels(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title)

        hbox = wx.BoxSizer(wx.HORIZONTAL)
        splitter = wx.SplitterWindow(self, -1)

        vbox1 = wx.BoxSizer(wx.VERTICAL)
        panel1 = wx.Panel(splitter, -1)
        panel11 = wx.Panel(panel1, -1, size=(-1, 40))
        panel11.SetBackgroundColour('#53728c')
        st1 = wx.StaticText(panel11, -1, 'Feeds', (5, 5))
        st1.SetForegroundColour('WHITE')

        panel12 = wx.Panel(panel1, -1, style=wx.BORDER_SUNKEN)
        panel12.SetBackgroundColour('WHITE')

        vbox1.Add(panel11, 0, wx.EXPAND)
        vbox1.Add(panel12, 1, wx.EXPAND)

        panel1.SetSizer(vbox1)

        vbox2 = wx.BoxSizer(wx.VERTICAL)
        panel2 = wx.Panel(splitter, -1)
        panel21 = wx.Panel(panel2, -1, size=(-1, 40), style=wx.NO_BORDER)
        st2 = wx.StaticText(panel21, -1, 'Articles', (5, 5))
        st2.SetForegroundColour('WHITE')

        panel21.SetBackgroundColour('#53728c')
        panel22 = wx.Panel(panel2, -1, style=wx.BORDER_RAISED)

        panel22.SetBackgroundColour('WHITE')
```

```
        vbox2.Add(panel21, 0, wx.EXPAND)
        vbox2.Add(panel22, 1, wx.EXPAND)

        panel2.SetSizer(vbox2)

        toolbar = self.CreateToolBar()
        toolbar.AddLabelTool(1, 'Exit', wx.Bitmap('icons/stock_exit.png'))
        toolbar.Realize()
        self.Bind(wx.EVT_TOOL, self.ExitApp, id=1)

        hbox.Add(splitter, 1, wx.EXPAND | wx.TOP | wx.BOTTOM, 5)
        self.SetSizer(hbox)
        self.CreateStatusBar()
        splitter.SplitVertically(panel1, panel2)
        self.Centre()
        self.Show(True)


    def ExitApp(self, event):
        self.Close()


app = wx.App()
Panels(None, -1, 'Panels')
app.MainLoop()
```

```
hbox = wx.BoxSizer(wx.HORIZONTAL)
splitter = wx.SplitterWindow(self, -1)
```

El *wx.SplitterWindow* dividirá nuestras ventanas dentro de dos partes.

Un panel podrá ser colocado sobre la izquierda y uno sobre el lado derecho. Cada uno tendrá otros dos paneles. Uno se creará una cabecera y los otros tomarán hasta el resto del panel padre. Juntos podremos usar seis paneles.

```
panel11 = wx.Panel(panel1, -1, size=(-1, 40))
panel11.SetBackgroundColour('#53728c')
st1 = wx.StaticText(panel11, -1, 'Feeds', (5, 5))
st1.SetForegroundColour('WHITE')
...
vbox1.Add(panel11, 0, wx.EXPAND)
```

Aquí creamos el panel cabecera. La altura de la cabecera es 40px. El color es fijado de color azul oscuro. (#53728c) Ponemos un *wx.StaticText* dentro del panel de cabecera. La posición es 5px desde la izquierda y 5px desde arriba de modo que tengamos algo de espacio entre el panel y el texto estático. El color del texto estático es fijado a blanco. Al final, hacemos el panel11 expandible y fijamos la proporción a 0.

```
panel12 = wx.Panel(panel1, -1, style=wx.BORDER_SUNKEN)
panel12.SetBackgroundColour('WHITE')
...
vbox1.Add(panel12, 1, wx.EXPAND)
```

El panel del fondo es creado con el estilo *wx.BORDER_SUNKEN*. El color es fijado a blanco. Lo hacemos expandible y fijamos el parámetro proporción a 1.

esta parte del tutorial de wxPython fue dedicada a componentes del núcleo de wxPython.

Componentes avanzados en wxPython

En los siguientes capítulos podremos hablar acerca de componentes avanzados. Una gran ventaja de wxPython sobre su competidor PyGTK es el disponibilidad de una gran cantidad de componentes avanzados. PyGTK es una capa sobre un kit de herramientas basado en C GKT+. Éste no provee nuevos componentes. En contraste, wxPython es una capa sobre wxWidgets que es un kit de herramientas basado en C++. wxWidgets consiste de las un grupo grande de componentes. Todos estos componentes son creados en C++. wxPython es un pegamento que combina lenguaje python con este kit de herramientas. Si buscamos tener un componente grilla en nuestras aplicaciones usando PyGTK, tenemos que crear lo nosotros mismos. Tal componente es bastante complicado. Por no hablar de la penalización de velocidad. Lenguajes dinámicos tales como Python, PERL o Ruby no son adecuado para tales tareas.

Los lenguajes dinámicos son grandes en varios areas. Aquellos son simples para usar. Aquellos son grandes para prototipado, en house desarrollo o para estudiar programa de computadora. Si necesitamos una solución rápida o necesitamos una aplicación, que cambiará rapidamente sobre un período corto de tiempo, los lenguajes dinámicos son superiores a los lenguajes compilados. De otra manera, si desarrollamos aplicaciones uso intensivo de recursos, juegos, aplicaciones multimedia de alta calidad, éstos no pueden competir con C++.

wxPython tiene varios componentes avanzados bien conocidos. Por ejemplo un componente árbol, una ventana html, un componente grilla, un componente caja de lista, un componente lista o un editor con avanzados capacidades de diseño. wxPython y wxWidgets están siendo desarrollados todos a un tiempo. Nuevos componentes y características emergen con cada gran lanzamiento. Al momento cuando escribo estas palabras un wxPython 2.8.3.0 será liberado justo en dos días. (22-Mar-2007).

A wx.ListBox componente

Un componente *wx.ListBox* es usado para mostrar y trabajar con una lista de items. Como su nombre lo indica, esto es un rectángulo que tiene una lista de cadenas por dentro. Podríamos usar lo para mostrar una lista de archivos mp3, nombres de libros, nombres de módulos de un gran proyecto o nombres de nuestros amigos. Un *wx.ListBox* puede ser creado en dos estados diferentes. En una simple estado de selección o un estado de selección múltiple. El estado de selección simple es el estado por omisión. Estos son dos eventos significativos en *wx.ListBox*. El primero es el evento *wx.EVT_COMMAND_LISTBOX_SELECTED*. Este evento es generado cuando seleccionamos una cadena en un *wx.ListBox*. El segundo es el evento *wx.EVT_COMMAND_LISTBOX_DOUBLE_CLICKED*. Éste es generado cuando hacemos doble clic en un item en a *wx.ListBox*. El número de elementos en el interior de un wx.ListBox está limitado sobre la plataforma GTK. De acuerdo a la documentación, éste es en la actualidad alrededor de 2000 elementos. Más que suficiente, pienso. Los elementos son numerados desde cero. Barras de deslizamiento son mostrada automáticamente si es necesario.

El constructor de un componente wx.ListBox es el siguiente:

```
wx.ListBox(wx.Window parent, int id=-1, wx.Point pos=wx.DefaultPosition,
wx.Size size=wx.DefaultSize,
        list choices=[], long style=0, wx.Validator
validator=wx.DefaultValidator,
        string name=wx.ListBoxNameStr)
```

Éste es un parámetro de elecciones. Si ponemos algunos valores aquí, aquellos se mostrarían desde el constructor del componente. Este parámetro esta vacio por omisión.

En nuestro código de ejemplo tenemos una caja de lista y cuatro botones. Cada uno de ellos llama diferentes métodos de nuestra caja de lista. Si buscamos agregar un nuevo item, llamamos el método *Append()*. Si buscamos borrar un item, llamamos el método *Delete()*. Para limpiar todas las cadenas en una caja de lista, llamamos el método Clear().

```
#!/usr/bin/python
# listbox.py

import wx

ID_NEW = 1
ID_RENAME = 2
ID_CLEAR = 3
ID_DELETE = 4


class ListBox(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(350, 220))

        panel = wx.Panel(self, -1)
        hbox = wx.BoxSizer(wx.HORIZONTAL)

        self.listbox = wx.ListBox(panel, -1)
        hbox.Add(self.listbox, 1, wx.EXPAND | wx.ALL, 20)

        btnPanel = wx.Panel(panel, -1)
        vbox = wx.BoxSizer(wx.VERTICAL)
        new = wx.button(btnPanel, ID_NEW, 'New', size=(90, 30))
        ren = wx.button(btnPanel, ID_RENAME, 'Rename', size=(90, 30))
        dlt = wx.button(btnPanel, ID_DELETE, 'Delete', size=(90, 30))
        clr = wx.button(btnPanel, ID_CLEAR, 'Clear', size=(90, 30))

        self.Bind(wx.EVT_BUTTON, self.NewItem, id=ID_NEW)
        self.Bind(wx.EVT_BUTTON, self.OnRename, id=ID_RENAME)
        self.Bind(wx.EVT_BUTTON, self.OnDelete, id=ID_DELETE)
        self.Bind(wx.EVT_BUTTON, self.OnClear, id=ID_CLEAR)
        self.Bind(wx.EVT_LISTBOX_DCLICK, self.OnRename)

        vbox.Add((-1, 20))
        vbox.Add(new)
        vbox.Add(ren, 0, wx.TOP, 5)
        vbox.Add(dlt, 0, wx.TOP, 5)
        vbox.Add(clr, 0, wx.TOP, 5)

        btnPanel.SetSizer(vbox)
        hbox.Add(btnPanel, 0.6, wx.EXPAND | wx.RIGHT, 20)
        panel.SetSizer(hbox)

        self.Centre()
        self.Show(True)

    def NewItem(self, event):
        text = wx.GetTextFromUser('Enter a new item', 'Insert dialog')
```

```
        if text != '':
            self.listbox.Append(text)

    def OnRename(self, event):
        sel = self.listbox.GetSelection()
        text = self.listbox.GetString(sel)
        renamed = wx.GetTextFromUser('Rename item', 'Rename dialog', text)
        if renamed != '':
            self.listbox.Delete(sel)
            self.listbox.Insert(renamed, sel)


    def OnDelete(self, event):
        sel = self.listbox.GetSelection()
        if sel != -1:
            self.listbox.Delete(sel)

    def OnClear(self, event):
        self.listbox.Clear()


app = wx.App()
ListBox(None, -1, 'ListBox')
app.MainLoop()
```

```
 self.listbox = wx.ListBox(panel, -1)
 hbox.Add(self.listbox, 1, wx.EXPAND | wx.ALL, 20)
```

Creamos una *wx.ListBox vacia*. Ponemos a 20px border alrededor de la caja de lista.

```
self.Bind(wx.EVT_LISTBOX_DCLICK, self.OnRename)
```

Vinculamos un tipo de evento *wx.EVT_COMMAND_LISTBOX_DOUBLE_CLICKED* con el método *OnRename()* usando el evento enlazador *wx.EVT_LISTBOX_DCLICK*. De esta manera mostraremos un diálogo de renombrado si hacemos doble clic sobre un elemento específico en la caja de lista.

```
 def NewItem(self, event):
     text = wx.GetTextFromUser('Enter a new item', 'Insert dialog')
     if text != '':
         self.listbox.Append(text)
```

Llamamos al método *NewItem()* haciendo clic en el botón New. Este método muestra una ventana de diálogo *wx.GetTextFromUser*. El texto que entramos es retornado a la variable de texto. Si el texto no esta vacio, lo agregamos al listbox con el método *Append()*.

```
def OnDelete(self, event):
    sel = self.listbox.GetSelection()
    if sel != -1:
        self.listbox.Delete(sel)
```

El borrado de un item es hecho en dos pasos. Primero encontramos el index del item seleccionado al llamar el método *GetSelection()*. Entonces borramos el item con el método *Delete()*. El parámetro al método *Delete()* es el índice seleccionado.

```
self.listbox.Delete(sel)
self.listbox.Insert(renamed, sel)
```

Aviso, ¿Cómo nos manejamos para renombrar una cadena?. El componente *wx.ListBox* no tiene método Rename(). Hicimos esta funcionalidad borrando la cadena previamente seleccionada e insertando una cadena nueva dentro de la posición del predecesor.

```
def OnClear(self, event):
    self.listbox.Clear()
```

Lo más fácil es para limpiar el cuadro de lista entero. Simplemente llamamos al método *Clear()*.



Un componente wx.ListBox

A wx.html.HtmlWindow componente

El componente *wx.html.HtmlWindow* muestra páginas html. Éste no es un navegador de pleno derecho. Podemos hacer interesantes cosas con el componente *wx.html.HtmlWindow*.

Formateado Especial

Por ejemplo en el siguiente guión vamos a crear una ventana, que muestra estadísticas básicas. Este formateo podría haber sido muy duro o casi imposible para crear sin el componente *wx.html.HtmlWindow*.

```
#!/usr/bin/python
import wx
import wx.html as html

ID_CLOSE = 1

page = '<html><body bgcolor="#8e8e95"><table celdaspacing="5" border="0"
width="250"> \
<tr width="200" align="left"> \
<td bgcolor="#e7e7e7">  Maximum</td> \
<td bgcolor="#aaaaaa">  <b>9000</b></td> \
</tr> \
<tr align="left"> \
<td bgcolor="#e7e7e7">  Mean</td> \
<td bgcolor="#aaaaaa">  <b>6076</b></td> \
</tr> \
<tr align="left"> \
<td bgcolor="#e7e7e7">  Minimum</td> \
<td bgcolor="#aaaaaa">  <b>3800</b></td> \
</tr> \
<tr align="left"> \
<td bgcolor="#e7e7e7">  Median</td> \
<td bgcolor="#aaaaaa">  <b>6000</b></td> \
```

```
</tr> \
<tr align="left"> \
<td bgcolor="#e7e7e7">  Standard Deviation</td> \
<td bgcolor="#aaaaaa">  <b>6076</b></td> \
</tr> \
</body></table></html>'


class MyFrame(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(400, 290))

        panel = wx.Panel(self, -1)

        vbox = wx.BoxSizer(wx.VERTICAL)
        hbox = wx.BoxSizer(wx.HORIZONTAL)

        htmlwin = html.HtmlWindow(panel, -1, style=wx.NO_BORDER)
        htmlwin.SetBackgroundColour(wx.RED)
        htmlwin.SetStandardFonts()
        htmlwin.SetPage(page)

        vbox.Add((-1, 10), 0)
        vbox.Add(htmlwin, 1, wx.EXPAND | wx.ALL, 9)

        bitmap = wx.StaticBitmap(panel, -1, wx.Bitmap('images/newt.png'))
        hbox.Add(bitmap, 1, wx.LEFT | wx.BOTTOM | wx.TOP, 10)
        buttonOk = wx.button(panel, ID_CLOSE, 'Ok')

        self.Bind(wx.EVT_BUTTON, self.OnClose, id=ID_CLOSE)

        hbox.Add((100, -1), 1, wx.EXPAND | wx.ALIGN_RIGHT)
        hbox.Add(buttonOk, flag=wx.TOP | wx.BOTTOM | wx.RIGHT, border=10)
        vbox.Add(hbox, 0, wx.EXPAND)

        panel.SetSizer(vbox)
        self.Centre()
        self.Show(True)

    def OnClose(self, event):
        self.Close()

app = wx.App(0)
MyFrame(None, -1, 'estadísticas básicas')
app.MainLoop()
```

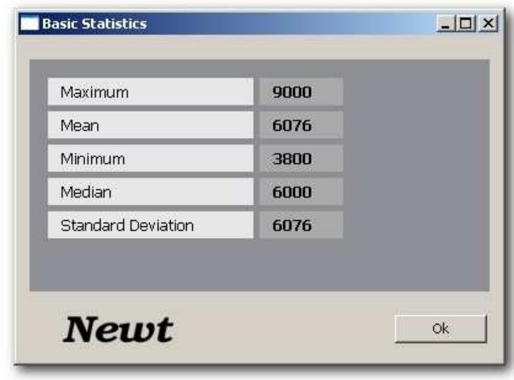


Figura: Ejemplo de ventana Html

Ventana de Ayuda

Nosotros podemos usar *wx.html.HtmlWindow* para proveer ayuda en nuestras aplicaciones. Podemos crear una ventana independiente o podemos crear una ventana, que está yendo a ser una parte de las aplicaciones. En el siguiente guión se creará una Ventana de Ayuda usando la última idea.

```
#!/usr/bin/python
# helpwindow.py

import wx
import wx.html as html

class HelpWindow(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(570, 400))

        toolbar = self.CreateToolBar()
        toolbar.AddLabelTool(1, 'Exit', wx.Bitmap('icons/exit.png'))
        toolbar.AddLabelTool(2, 'Help', wx.Bitmap('icons/help.png'))
        toolbar.Realize()

        self.splitter = wx.SplitterWindow(self, -1)
        self.panelLeft = wx.Panel(self.splitter, -1, style=wx.BORDER_SUNKEN)

        self.panelRight = wx.Panel(self.splitter, -1)
        vbox2 = wx.BoxSizer(wx.VERTICAL)
        cabecera = wx.Panel(self.panelRight, -1, size=(-1, 20))
        header.SetBackgroundColour('#6f6a59')
        header.SetForegroundColour('WHITE')
        hbox = wx.BoxSizer(wx.HORIZONTAL)

        st = wx.StaticText(header, -1, 'Help', (5, 5))
        font = st.GetFont()
        font.SetPointSize(9)
        st.SetFont(font)
        hbox.Add(st, 1, wx.TOP | wx.BOTTOM | wx.LEFT, 5)

        close = wx.BitmapButton(header, -1, wx.Bitmap('icons/fileclose.png',
wx.BITMAP_TYPE_PNG),
                style=wx.NO_BORDER)
        close.SetBackgroundColour('#6f6a59')
        hbox.Add(close, 0)
        header.SetSizer(hbox)

        vbox2.Add(header, 0, wx.EXPAND)

        help = html.HtmlWindow(self.panelRight, -1, style=wx.NO_BORDER)
        help.LoadPage('help.html')
        vbox2.Add(help, 1, wx.EXPAND)
        self.panelRight.SetSizer(vbox2)
        self.panelLeft.SetFocus()

        self.splitter.SplitVertically(self.panelLeft, self.panelRight)
```

```
        self.splitter.Unsplit()

        self.Bind(wx.EVT_BUTTON, self.CloseHelp, id=close.GetId())
        self.Bind(wx.EVT_TOOL, self.OnClose, id=1)
        self.Bind(wx.EVT_TOOL, self.OnHelp, id=2)

        self.Bind(wx.EVT_KEY_DOWN, self.OnKeyPressed)

        self.CreateStatusBar()

        self.Centre()
        self.Show(True)

    def OnClose(self, event):
        self.Close()

    def OnHelp(self, event):
        self.splitter.SplitVertically(self.panelLeft, self.panelRight)
        self.panelLeft.SetFocus()

    def CloseHelp(self, event):
        self.splitter.Unsplit()
        self.panelLeft.SetFocus()

    def OnKeyPressed(self, event):
        keycode = event.GetKeyCode()
        if keycode == wx.WXK_F1:
            self.splitter.SplitVertically(self.panelLeft, self.panelRight)
            self.panelLeft.SetFocus()


app = wx.App()
HelpWindow(None, -1, 'HelpWindow')
app.MainLoop()
```

La Ventana de Ayuda esta oculta al comienzo. Podemos mostrarla haciendo clic en el bóton de ayuda sobre la barra de herramientas o por presionando F1. La Ventana de Ayuda aparece sobre el lado derecho de las aplicaciones. Para ocultar Ventana de Ayuda, hacemos clic sobre el botón close.

```
 self.splitter.SplitVertically(self.panelLeft, self.panelRight)
 self.splitter.Unsplit()
```

Creamos paneles izquierdos y derechos y los dividimos verticalmente. Después de esto, llamamos el método *Unsplit()*. Por omisión el método oculta los paneles derechos o del fondo.

Dividimos el panel derecho dentro de dos partes. La cabecera y el cuerpo del panel. La cabecera es un *wx.Panel ajustado*. La cabecera consiste tanto de un texto estático y como botón bitmap. Ponemos un *wx.html.Window* dentro del cuerpo del panel.

```
 close = wx.BitmapButton(header, -1, wx.Bitmap('icons/fileclose.png',
wx.BITMAP_TYPE_PNG),
        style=wx.NO_BORDER)
 close.SetBackgroundColour('#6f6a59')
```

El estilo del botón bitmap es fijado a *wx.NO_BORDER*. El color de fondo es fijado al color del panel de cabecera. Éste es hecho en orden de hacer el botón aparezca como una parte de la cabecera.

```
help = html.HtmlWindow(self.panelRight, -1, style=wx.NO_BORDER)
help.LoadPage('help.html')
```

Creamos un componente wx.html.HtmlWindow sobre el el panel derecho. Tenemos nuestras código HTML en un archivo separado. En este momento llamamos el método*LoadPage()* para obtener el código HTML.

```
self.panelLeft.SetFocus()
```

Fijamos el foco sobre el panel de la izquierda. Podemos lanzar la Ventana de Ayuda con la tecla F1. A los efectos de controlar una ventana con un teclado, éste debe tener el foco. Si no fijamos el foco, deberíamos hacer el primer clic sobre el panel y solamente entonces podríamos lanzar la Ventana de Ayuda presionando la tecla F1.

```
    def OnHelp(self, event):
        self.splitter.SplitVertically(self.panelLeft, self.panelRight)
        self.panelLeft.SetFocus()
```

Para mostrar la Ventana de Ayuda, llamamos al método *OnHelp()*. Éste divide los dos paneles verticalmente. No debemos olvidar fijar el foco de nuevo, porque el foco inicial se pierde al dividir.

El siguientes es el archivo html, que cargamos en nuestras aplicaciones.

```
<html>

<body bgcolor="#ababab">
<h4>Table of Contents</h4>

<ul>
<li><a href="#basic">estadísticas básicas</a></li>
<li><a href="#advanced">Advanced statistics</a></li>
<li><a href="#intro">Introducing Newt</a></li>
<li><a href="#charts">work with charts</a></li>
<li><a href="#pred">Predicting values</a></li>
<li><a href="#neural">Neural networks</a></li>
<li><a href="#glos">Glossary</a></li>
</ul>

<p>
<a name="basic">
<h6>basic statics</h6>
Overview delementary concepts in statistics.
Variables. Correlation. Measurement scales. Statistical significance.
Distributions. Normality assumption.
</a>
</p>

<p>
<a name="advanced">
<h6>Advanced Statistics</h6>
Overview of advanced concepts in statistics. Anova. Linear regression.
Estimation y  hypothesis testing.
Error terms.
</a>
</p>

<p>
```

```
<a name="intro">
<h6>Introducing Newt</h6>
Introducing the functionality basic of the Newt application. Creating
sheets.
Charts. Menus y Toolbars. Importing data. Saving data en varios formats.
Exporting data. Shortcuts. List de methods.
</a>
</p>

<p>
<a name="charts">
<h6>Charts</h6>
work with charts. 2D charts. 3D charts. Bar, line, box, pie, range charts.
Scatterplots. Histograms.
</a>
</p>

<p>
<a name="pred">
<h6>Predicting values</h6>
Time series y forecasting. Trend Analysis. Seasonality. Moving averages.
Univariate métodos. Multivariate methods. Holt-Winters smoothing.
Exponential smoothing. ARIMA. cuatroier analysis.
</a>
</p>

<p>
<a name="neural">
<h6>Neural networks</h6>
Overview de neural networks. biología behind neural networks.
Basic artificial Model. Training. Preprocesamiento. Postprocesamiento.
Tipos de neural networks.
</a>
</p>

<p>
<a name="glos">
<h6>Glossary</h6>
Terms y definitions en statistics.
</a>
</p>

</body>
</html>
```

```
 <li><a href="#basic">estadísticas básicas</a></li>
 ...
 <a name="basic">
```

Normalmente podría escribir <div id="basic"> ... </div>. Ambas son anotaciones HTML correctas. Pero *wx.html.HtmlWindow* soporta solamente el primero. *wx.html.HtmlWindow* soporta solamente un subconjunto del lenguaje de marcado HTML.



Figura: Ventana de Ayuda

Un componentewx.ListCtrl

Un*wx.ListCtrl* es una representación gráfica de una lista de items. Un *wx.ListBox* puede solamente tener una columna. *wx.ListCtrl* puede tener más que una columna. *wx.ListCtrl* es un componente muy común y útil. Por ejemplo un administrador de archivos usa un *wx.ListCtrl* para mostrar directorios y archivos sobre el sistema de archivos. Una aplicación "quemador de CD" muestra archivos para ser quemados en el interior de un *wx.ListCtrl*.

Un *wx.ListCtrl* puede ser usado con tres parámetros diferentes. En una vista de lista, vista de reporte o una vista de iconos. Estos formatos son controlados por los estilos de ventana del *wx.ListCtrl*. *wx.LC_REPORT*, *wx.LC_LIST* y *wx.LC_ICON*.

```
 wx.ListCtrl(wx.Window parent, int id, wx.Point pos = (-1, -1), wx.Size size
= (-1, -1),
 int style = wx.LC_ICON, wx.Validator validator = wx.DefaultValidator,
string name = wx.ListCtrlNameStr)
```

Estilos wx.ListCtrl

- wx.LC_LIST
- wx.LC_REPORT
- wx.LC_VIRTUAL
- wx.LC_ICON
- wx.LC_SMALL_ICON
- wx.LC_ALIGN_LEFT
- wx.LC_EDIT_LABELS
- wx.LC_NO_HEADER
- wx.LC_SORT_ASCENDING
- wx.LC_SORT_DESCENDING
- wx.LC_HRULES
- wx.LC_VRULES

Ejemplo simple

En el primer ejemplo vamos a introducir la funcionalidad básica de un *wx.ListCtrl*.

```
#!/usr/bin/python
# actresses.py

import wx
import sys

packages = [('jessica alba', 'pomona', '1981'), ('sigourney weaver', 'new
york', '1949'),
    ('angelina jolie', 'los angeles', '1975'), ('natalie portman',
'jerusalem', '1981'),
    ('rachel weiss', 'london', '1971'), ('scarlett johansson', 'new york',
'1984' )]



class Actresses(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(380, 230))

        hbox = wx.BoxSizer(wx.HORIZONTAL)
        panel = wx.Panel(self, -1)

        self.list = wx.ListCtrl(panel, -1, style=wx.LC_REPORT)
        self.list.InsertColumn(0, 'name', width=140)
        self.list.InsertColumn(1, 'place', width=130)
        self.list.InsertColumn(2, 'year', wx.LIST_FORMAT_RIGHT, 90)

        for i in packages:
            index = self.list.InsertStringItem(sys.maxint, i[0])
            self.list.SetStringItem(index, 1, i[1])
            self.list.SetStringItem(index, 2, i[2])

        hbox.Add(self.list, 1, wx.EXPAND)
        panel.SetSizer(hbox)

        self.Centre()
        self.Show(True)

app = wx.App()
Actresses(None, -1, 'actresses')
app.MainLoop()
```

```
 self.list = wx.ListCtrl(panel, -1, style=wx.LC_REPORT)
```

creamos a *wx.ListCtrl* con a wx.LC_REPORT estilo.

```
 self.list.InsertColumn(0, 'name', width=140)
 self.list.InsertColumn(1, 'place', width=130)
 self.list.InsertColumn(2, 'year', wx.LIST_FORMAT_RIGHT, 90)
```

Insertamos tres columnas. Podemos especificar el ancho de la columna y el formato de la columna. El formato por omisión es *wx.LIST_FORMAT_LEFT*.

```
 for i in packages:
     index = self.list.InsertStringItem(sys.maxint, i[0])
     self.list.SetStringItem(index, 1, i[1])
     self.list.SetStringItem(index, 2, i[2])
```

Insertamos datos dentro del *wx.ListCtrl* usando dos métodos. Cada fila inicia con un método *InsertStringItem()*. El primer parámetro del método especifica el número de fila. Al dar un sys.maxint aseguramos, que cada llamada podrá insertar datos después de la última fila. El método retorna el índice de fila. El método *SetStringItem()* agrega datos a las columnas consecutivas de la fila actual.

Mixins

Mixins son clases que mejora ampliamente la funcionalidad de una *wx.ListCtrl*. Mixin clases son como llamadas a clases de ayuda. Aquellas están localizados en el módulo *wx.lib.mixins.listctrl*. En orden para usar ellos, el programador tiene que heredar desde estas clases.

Estos son cinco mixins disponibles. Desde 2.8.1.1.

- wx.ColumnSorterMixin
- wx.ListCtrlAutoWidthMixin
- wx.ListCtrlSelectionManagerMix
- wx.TextEditMixin
- wx.CheckListCtrlMixin

wx.ColumnSorterMixin es una mixin que permite clasificar las columnas en una vista de reporte. La clase *wx.ListCtrlAutoWidthMixin* automáticamente redimensiona la última columna al final del *wx.ListCtrl*. Por omisión, la última columna no toma el espacio remanente. Ver el ejemplo previo. *wx.ListCtrlSelectionManagerMix* define política de selección de plataforma independiente. *wx.TextEditMixin* permite texto para ser editado. *wx.CheckListCtrlMixin* agrega una caja de verificación a cada fila. De esta manera podemos controlar filas. Podemos fijar cada fila para ser verificada o desverificada.

El código siguiente muestra, como podemos usar *ListCtrlAutoWidthMixin*

```
#!/usr/bin/python
# autowidth.py

import wx
import sys
from wx.lib.mixins.listctrl import ListCtrlAutoWidthMixin

actresses = [('jessica alba', 'pomona', '1981'), ('sigourney weaver', 'new
york', '1949'),
    ('angelina jolie', 'los angeles', '1975'), ('natalie portman',
'jerusalem', '1981'),
    ('rachel weiss', 'london', '1971'), ('scarlett johansson', 'new york',
'1984' )]


class AutoWidthListCtrl(wx.ListCtrl, ListCtrlAutoWidthMixin):
    def __init__(self, parent):
        wx.ListCtrl.__init__(self, parent, -1, style=wx.LC_REPORT)
        ListCtrlAutoWidthMixin.__init__(self)


class Actresses(wx.Frame):
```

```
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(380, 230))

        hbox = wx.BoxSizer(wx.HORIZONTAL)

        panel = wx.Panel(self, -1)

        self.list = AutoWidthListCtrl(panel)
        self.list.InsertColumn(0, 'name', width=140)
        self.list.InsertColumn(1, 'place', width=130)
        self.list.InsertColumn(2, 'year', wx.LIST_FORMAT_RIGHT, 90)

        for i in actresses:
            index = self.list.InsertStringItem(sys.maxint, i[0])
            self.list.SetStringItem(index, 1, i[1])
            self.list.SetStringItem(index, 2, i[2])

        hbox.Add(self.list, 1, wx.EXPAND)
        panel.SetSizer(hbox)

        self.Centre()
        self.Show(True)

app = wx.App()
Actresses(None, -1, 'actresses')
app.MainLoop()
```

Cambiamos el ejemplo previo un poquito.

```
from wx.lib.mixins.listctrl import ListCtrlAutoWidthMixin
```

Aquí importamos el mixin.

```
class AutoWidthListCtrl(wx.ListCtrl, ListCtrlAutoWidthMixin):
    def __init__(self, parent):
        wx.ListCtrl.__init__(self, parent, -1, style=wx.LC_REPORT)
        ListCtrlAutoWidthMixin.__init__(self)
```

Creamos una nueva clase *AutoWidthListCtrl*. Esta clase herederará desde *wx.ListCtrl* y *ListCtrlAutoWidthMixin*. Ésto es llamado **Herencia** múltiple. La última columna podrá redimensionarse automáticamente para tomar todo el ancho remanente de un *wx.ListCtrl*.

Figura: ejemplo AutoWidth



En el siguientes ejemplo mostraremos, como podemos crear columnas clasificables. Si hacemos clic sobre la cabecera de la columna, las filas correspondientes en una columna son clasificadas.

```
#!/usr/bin/python
# sorted.py

import wx
import sys
from wx.lib.mixins.listctrl import ColumnSorterMixin

actresses = {
1 : ('jessica alba', 'pomona', '1981'),
2 : ('sigourney weaver', 'new york', '1949'),
3 : ('angelina jolie', 'los angeles', '1975'),
4 : ('natalie portman', 'jerusalem', '1981'),
5 : ('rachel weiss', 'london', '1971'),
6 : ('scarlett johansson', 'new york', '1984')
}


class SortedListCtrl(wx.ListCtrl, ColumnSorterMixin):
    def __init__(self, parent):
        wx.ListCtrl.__init__(self, parent, -1, style=wx.LC_REPORT)
        ColumnSorterMixin.__init__(self, len(actresses))
        self.itemDataMap = actresses

    def GetListCtrl(self):
        return self

class Actresses(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(380, 230))

        hbox = wx.BoxSizer(wx.HORIZONTAL)

        panel = wx.Panel(self, -1)

        self.list = SortedListCtrl(panel)
        self.list.InsertColumn(0, 'name', width=140)
        self.list.InsertColumn(1, 'place', width=130)
        self.list.InsertColumn(2, 'year', wx.LIST_FORMAT_RIGHT, 90)

        items = actresses.items()

        for key, data en items:
            index = self.list.InsertStringItem(sys.maxint, data[0])
            self.list.SetStringItem(index, 1, data[1])
            self.list.SetStringItem(index, 2, data[2])
            self.list.SetItemData(index, key)

        hbox.Add(self.list, 1, wx.EXPAND)
        panel.SetSizer(hbox)

        self.Centre()
        self.Show(True)
```

```
app = wx.App()
Actresses(None, -1, 'actresses')
app.MainLoop()
```

Podríamos de nuevo usar el ejemplo con actrices.

```
ColumnSorterMixin.__init__(self, len(actresses))
```

El *ColumnSorterMixin* acepta un argumento. Éste es el número de columnas para ser clasificadas.

```
self.itemDataMap = actresses
```

Debemos mapear nuestros datos para ser mostrados en una lista de control al atributo *itemDataMap*. El dato debe estar en un tipo de dato diccionario.

```
def GetListCtrl(self):
    return self
```

Debemos crear un método *GetListCtrl()*. Este método retorna el componente *wx.ListCtrl* que está yendo para ser clasificado.

```
 self.list.SetItemData(index, key)
```

Debemos asociar cada fila con un índice especial. Ésto es hecho con el método *SetItemData*.

Reader

Un reader es un ejemplo complejo mostrando dos listas de controles en una vista de reporte.

```
#!/usr/bin/python
# reader.py

import wx

articles = [['Mozilla rocks', 'El year del Mozilla', 'Earth on Fire'],
            ['Gnome pretty, Gnome Slow', 'Gnome, KDE, Icewm, XFCE', 'WEste 
es Gnome heading?'],
            ['Java number uno lenguaje', 'Compiled languages, intrepreted 
Languages', 'Java on Desktop?']]



class ListCtrlLeft(wx.ListCtrl):
    def __init__(self, parent, id):
        wx.ListCtrl.__init__(self, parent, id, style=wx.LC_REPORT | 
wx.LC_HRULES |
                wx.LC_NO_HEADER | wx.LC_SINGLE_SEL)
        images = ['icons/java.png', 'icons/gnome.png', 'icons/mozilla.png']

        self.parent = parent

        self.Bind(wx.EVT_SIZE, self.OnSize)
        self.Bind(wx.EVT_LIST_ITEM_SELECTED, self.OnSelect)

        self.il = wx.ImageList(32, 32)
        for i in images:
            self.il.Add(wx.Bitmap(i))
```

```
        self.SetImageList(self.il, wx.IMAGE_LIST_SMALL)
        self.InsertColumn(0, '')

        for i in range(3):
            self.InsertStringItem(0, '')
            self.SetItemImage(0, i)

    def OnSize(self, event):
        size = self.parent.GetSize()
        self.SetColumnWidth(0, size.x-5)
        event.Skip()

    def OnSelect(self, event):
        window =
self.parent.GetGrandParent().FindWindowByName('ListControlOnRight')
        index = event.GetIndex()
        window.LoadData(index)

    def OnDeSelect(self, event):
        index = event.GetIndex()
        self.SetItemBackgroundColour(index, 'WHITE')

    def OnFocus(self, event):
        self.SetItemBackgroundColour(0, 'red')

class ListCtrlRight(wx.ListCtrl):
    def __init__(self, parent, id):
        wx.ListCtrl.__init__(self, parent, id, style=wx.LC_REPORT |
wx.LC_HRULES |
                wx.LC_NO_HEADER | wx.LC_SINGLE_SEL)

        self.parent = parent

        self.Bind(wx.EVT_SIZE, self.OnSize)

        self.InsertColumn(0, '')


    def OnSize(self, event):
        size = self.parent.GetSize()
        self.SetColumnWidth(0, size.x-5)
        event.Skip()

    def LoadData(self, index):
        self.DeleteAllItems()
        for i in range(3):
            self.InsertStringItem(0, articles[index][i])


class Reader(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title)

        hbox = wx.BoxSizer(wx.HORIZONTAL)
        splitter = wx.SplitterWindow(self, -1, style=wx.SP_LIVE_UPDATE|
```

```
wx.SP_NOBORDER)

        vbox1 = wx.BoxSizer(wx.VERTICAL)
        panel1 = wx.Panel(splitter, -1)
        panel11 = wx.Panel(panel1, -1, size=(-1, 40))
        panel11.SetBackgroundColour('#53728c')
        st1 = wx.StaticText(panel11, -1, 'Feeds', (5, 5))
        st1.SetForegroundColour('WHITE')

        panel12 = wx.Panel(panel1, -1, style=wx.BORDER_SUNKEN)
        vbox = wx.BoxSizer(wx.VERTICAL)
        list1 = ListCtrlLeft(panel12, -1)

        vbox.Add(list1, 1, wx.EXPAND)
        panel12.SetSizer(vbox)
        panel12.SetBackgroundColour('WHITE')


        vbox1.Add(panel11, 0, wx.EXPAND)
        vbox1.Add(panel12, 1, wx.EXPAND)

        panel1.SetSizer(vbox1)

        vbox2 = wx.BoxSizer(wx.VERTICAL)
        panel2 = wx.Panel(splitter, -1)
        panel21 = wx.Panel(panel2, -1, size=(-1, 40), style=wx.NO_BORDER)
        st2 = wx.StaticText(panel21, -1, 'Articles', (5, 5))
        st2.SetForegroundColour('WHITE')

        panel21.SetBackgroundColour('#53728c')
        panel22 = wx.Panel(panel2, -1, style=wx.BORDER_RAISED)
        vbox3 = wx.BoxSizer(wx.VERTICAL)
        list2 = ListCtrlRight(panel22, -1)
        list2.SetName('ListControlOnRight')
        vbox3.Add(list2, 1, wx.EXPAND)
        panel22.SetSizer(vbox3)


        panel22.SetBackgroundColour('WHITE')
        vbox2.Add(panel21, 0, wx.EXPAND)
        vbox2.Add(panel22, 1, wx.EXPAND)

        panel2.SetSizer(vbox2)

        toolbar = self.CreateToolBar()
        toolbar.AddLabelTool(1, 'Exit', wx.Bitmap('icons/stock_exit.png'))
        toolbar.Realize()

        self.Bind(wx.EVT_TOOL, self.ExitApp, id=1)

        hbox.Add(splitter, 1, wx.EXPAND | wx.TOP | wx.BOTTOM, 5)
        self.SetSizer(hbox)
        self.CreateStatusBar()
        splitter.SplitVertically(panel1, panel2)
        self.Centre()
        self.Show(True)
```

```
    def ExitApp(self, event):
        self.Close()


app = wx.App()
Reader(None, -1, 'Reader')
app.MainLoop()
```

El ejemplo previo muestra un *wx.ListCtrl* en una vista de reporte. Sin cabeceras. Deberíamos crear nuestras propias cabeceras. Mostraremos dos componentes *wx.ListCtrl*. Uno es sobre el lado derecho y el otros sobre el lado izquierdo de la aplicación.

```
 splitter = wx.SplitterWindow(self, -1, style=wx.SP_LIVE_UPDATE|
wx.SP_NOBORDER)
 ...
 splitter.SplitVertically(panel1, panel2)
```

El splitter dividirá la ventana principal dentro de dos partes verticales. El splitter mostrarán dos paneles. Estos dos paneles tendrán otros dos paneles. Aquellos crean cabeceras *Feeds* y *Articles*. El resto del espacio podrá ser ocupado por nuestros componentes dos *wx.ListCtrl*.

```
 list2 = ListCtrlRight(panel22, -1)
 list2.SetName('ListControlOnRight')
```

Cuando creamos el objeto *ListCtrlRight*, le damos un nombre *ListControlOnRight*. Éste es porque necesitamos los dos componentes *ListCtrlRight* y *ListCtrlLeft* para comunicar.

```
 def OnSelect(self, event):
     window =
self.parent.GetGrandParent().FindWindowByName('ListControlOnRight')
     index = event.GetIndex()
     window.LoadData(index)
```

Este código está en la clase *ListCtrlLeft*. Aquí ubicamos el objeto *ListCtrlRight* y llamamos al método *LoadData() de él*.

```
 def LoadData(self, index):
     self.DeleteAllItems()
     for i in range(3):
         self.InsertStringItem(0, articles[index][i])
```

El método *LoadData()* primero limpia todos los items. Entonces este inserta el nombre del artículo desde la lista de artículos definidos globalmente. El índice tiene que ser pasado.

```
 def OnSize(self, event):
     size = self.parent.GetSize()
     self.SetColumnWidth(0, size.x-5)
     event.Skip()
```

Ambos *wx.ListCtrls* tienen solamente una columna. Aquí aseguramos que el tamaño de la columna sea igual al tamaño del panel padre. Las aplicaciones podrían no verse bien de otros modos. ¿Por qué extraemos 5px? Este número es un tipo de número mágico. Si extraemos exactamente 5px, la barra de deslizamiento horizontal no aparecen. Sobre otras plataformas de SO, el número sería diferente.



Figura: Reader

CheckListCtrl

Éste es de lo más común para ver aplicaciones teniendo casillas de verificación en el interior de una lista de controles. Por ejemplo aplicaciones empaquetadas tales como Synaptic o KYUM.

Dese el punto de vista del programador, estas casillas de verificación son simples imagenes. Estos tienen dos estados. Verificada y desverificada. Para estas situaciones tenemos una imagen única. No tenemos que implementar la funcionalidad. Éste tiene que ser siempre codificado. El código es en *CheckListCtrlMixin*.

```
#!/usr/bin/python
# repository.py

import wx
import sys
from wx.lib.mixins.listctrl import CheckListCtrlMixin,
ListCtrlAutoWidthMixin

packages = [('abiword', '5.8M', 'base'), ('adie', '145k', 'base'),
    ('airsnort', '71k', 'base'), ('ara', '717k', 'base'), ('arc', '139k',
'base'),
    ('asc', '5.8M', 'base'), ('ascii', '74k', 'base'), ('ash', '74k',
'base')]

class CheckListCtrl(wx.ListCtrl, CheckListCtrlMixin,
ListCtrlAutoWidthMixin):
    def __init__(self, parent):
        wx.ListCtrl.__init__(self, parent, -1, style=wx.LC_REPORT |
wx.SUNKEN_BORDER)
        CheckListCtrlMixin.__init__(self)
        ListCtrlAutoWidthMixin.__init__(self)


class Repository(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(450, 400))

        panel = wx.Panel(self, -1)

        vbox = wx.BoxSizer(wx.VERTICAL)
        hbox = wx.BoxSizer(wx.HORIZONTAL)
```

```
        leftPanel = wx.Panel(panel, -1)
        rightPanel = wx.Panel(panel, -1)

        self.log = wx.TextCtrl(rightPanel, -1, style=wx.TE_MULTILINE)
        self.list = CheckListCtrl(rightPanel)
        self.list.InsertColumn(0, 'Package', width=140)
        self.list.InsertColumn(1, 'Size')
        self.list.InsertColumn(2, 'Repository')

        for i in packages:
            index = self.list.InsertStringItem(sys.maxint, i[0])
            self.list.SetStringItem(index, 1, i[1])
            self.list.SetStringItem(index, 2, i[2])

        vbox2 = wx.BoxSizer(wx.VERTICAL)

        sel = wx.button(leftPanel, -1, 'Select All', size=(100, -1))
        des = wx.button(leftPanel, -1, 'Deselect All', size=(100, -1))
        apply = wx.button(leftPanel, -1, 'Apply', size=(100, -1))


        self.Bind(wx.EVT_BUTTON, self.OnSelectAll, id=sel.GetId())
        self.Bind(wx.EVT_BUTTON, self.OnDeselectAll, id=des.GetId())
        self.Bind(wx.EVT_BUTTON, self.OnApply, id=apply.GetId())

        vbox2.Add(sel, 0, wx.TOP, 5)
        vbox2.Add(des)
        vbox2.Add(apply)

        leftPanel.SetSizer(vbox2)

        vbox.Add(self.list, 1, wx.EXPAND | wx.TOP, 3)
        vbox.Add((-1, 10))
        vbox.Add(self.log, 0.5, wx.EXPAND)
        vbox.Add((-1, 10))

        rightPanel.SetSizer(vbox)

        hbox.Add(leftPanel, 0, wx.EXPAND | wx.RIGHT, 5)
        hbox.Add(rightPanel, 1, wx.EXPAND)
        hbox.Add((3, -1))

        panel.SetSizer(hbox)

        self.Centre()
        self.Show(True)

    def OnSelectAll(self, event):
        num = self.list.GetItemCount()
        for i in range(num):
            self.list.CheckItem(i)

    def OnDeselectAll(self, event):
        num = self.list.GetItemCount()
        for i in range(num):
            self.list.CheckItem(i, False)
```

```
    def OnApply(self, event):
        num = self.list.GetItemCount()
        for i in range(num):
            if i == 0: self.log.Clear()
            if self.list.IsChecked(i):
                self.log.AppendText(self.list.GetItemText(i) + '\n')

app = wx.App()
Repository(None, -1, 'Repository')
app.MainLoop()
```

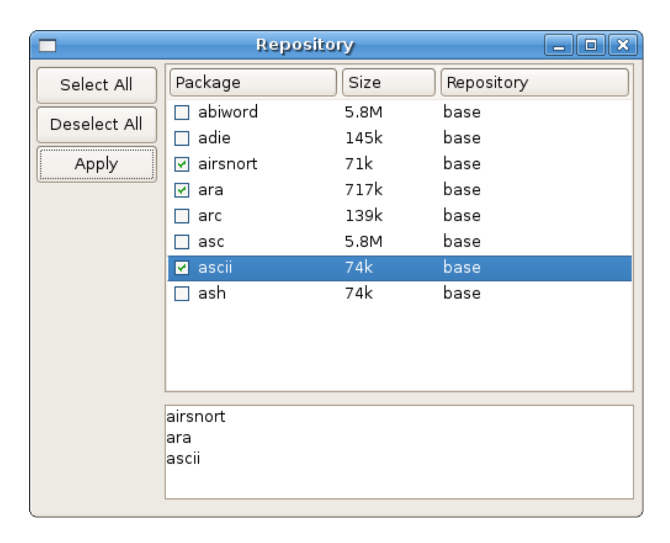


Figura: Repository

```
 class CheckListCtrl(wx.ListCtrl, CheckListCtrlMixin,
ListCtrlAutoWidthMixin):
     def __init__(self, parent):
         wx.ListCtrl.__init__(self, parent, -1, style=wx.LC_REPORT |
wx.SUNKEN_BORDER)
         CheckListCtrlMixin.__init__(self)
         ListCtrlAutoWidthMixin.__init__(self)
```

wxPython permite Herencia múltiple. Aquí heredamos desde tres diferentes clases.

```
 def OnSelectAll(self, event):
     num = self.list.GetItemCount()
     for i in range(num):
         self.list.CheckItem(i)
```

Aquí podemos ver Herencia múltiple en acción. Podemos llamar dos métodos desde dos diferentes clases sobre nuestro objeto *self.list*. El método *GetItemCount()* está localizada en la clase *CheckListCtrl* y el método *CheckItem()* está en la clase *CheckListCtrlMixin*.

En esta parte del tutorial de wxPython, cubrimos varios componentes avanzados.

Arrastrar y soltar en wxPython

Wikipedia: En interfaces gráficas del usuario de computadoras, arrastrar-y-soltar es la acción de (o soportar para la acción de) hacer clic sobre un objeto virtual y arrastrarlo a diferentes ubicaciones o sobre otro objeto virtual. En general, lo pueden usar para invocar muchas tipos de acciones, o crear varios tipos de asociaciones entre dos objetos abstractos.

La funcionalidad de arrastrar y soltar es uno de los aspectos más visibles del interfaz gráfica del usuario. La operación de arrastrar y soltar le permite a usted para hacer cosas complejas intuitivamente.

En arrastrar y soltar arrastramos basicamente algunos datos desde una fuente de datos un objetivo de datos. Así debemos tener:

- Algunos datos
- Una fuente de datos
- Un objetivo de datos

En wxPython tenemos dos objetivos de datos predefinidos. **wx.TextDropTarget** y **wx.FileDropTarget**.

- wx.TextDropTarget
- wx.FileDropTarget

wx.TextDropTarget

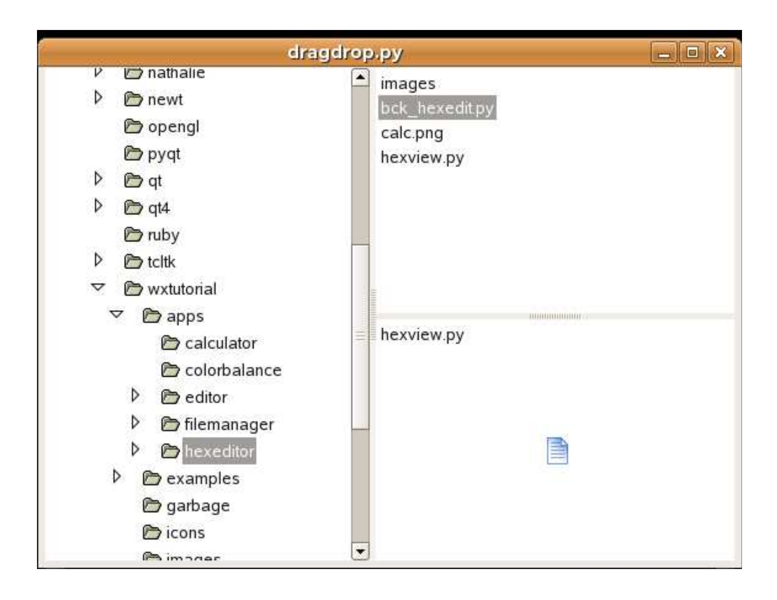


Figura: dragdrop.py

```
#!/usr/bin/python
# dragdrop.py

import os
import wx

class MyTextDropTarget(wx.TextDropTarget):
    def __init__(self, object):
        wx.TextDropTarget.__init__(self)
        self.object = object
```

```
    def OnDropText(self, x, y, data):
        self.object.InsertStringItem(0, data)


class DragDrop(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(650, 500))

        splitter1 = wx.SplitterWindow(self, -1, style=wx.SP_3D)
        splitter2 = wx.SplitterWindow(splitter1, -1, style=wx.SP_3D)
        self.dir = wx.GenericDirCtrl(splitter1, -1, dir='/home/',
style=wx.DIRCTRL_DIR_ONLY)
        self.lc1 = wx.ListCtrl(splitter2, -1, style=wx.LC_LIST)
        self.lc2 = wx.ListCtrl(splitter2, -1, style=wx.LC_LIST)

        dt = MyTextDropTarget(self.lc2)
        self.lc2.SetDropTarget(dt)
        self.Bind(wx.EVT_LIST_BEGIN_DRAG, self.OnDragInit,
id=self.lc1.GetId())

        tree = self.dir.GetTreeCtrl()

        splitter2.SplitHorizontally(self.lc1, self.lc2)
        splitter1.SplitVertically(self.dir, splitter2)

        self.Bind(wx.EVT_TREE_SEL_CHANGED, self.OnSelect, id=tree.GetId())

        self.OnSelect(0)
        self.Centre()
        self.Show(True)

    def OnSelect(self, event):
        list = os.listdir(self.dir.GetPath())
        self.lc1.ClearAll()
        self.lc2.ClearAll()
        for i in range(len(list)):
            if list[i][0] != '.':
                self.lc1.InsertStringItem(0, list[i])

    def OnDragInit(self, event):
        text = self.lc1.GetItemText(event.GetIndex())
        tdo = wx.TextDataObject(text)
        tds = wx.DropSource(self.lc1)
        tds.SetData(tdo)
        tds.DoDragDrop(True)


app = wx.App()
DragDrop(None, -1, 'dragdrop.py')
app.MainLoop()
```

wx.FileDropTarget

Una de las ventajas del GUI sobre la consola es es la intuición. Usted puede aprender un programa de

interfaz gráfica de usuario más fácil que una aplicación de consola. A menudo no es necesario un manual. Por otro lado, algunas operaciones gráficas son demasiado complejas. Por ejemplo, el borrado de un archivo por arrastrarlo y soltarlo en la papelera es muy intuitivo y fácil de entender, pero en realidad la mayoría de la gente solo pulsa la tecla delete. (shift + delete) esto es más efectivo. En nuestro ejemplo siguiente exploramos una operación gráfica, que es muy práctica. En la mayoría de los editores de texto GUI, usted puede abrir un archivo por simplemente arrastrarlo desde el administrador de archivos y soltandolo sobre el editor.

```
#!/usr/bin/python
# filedrop.py

import wx

class FileDrop(wx.FileDropTarget):
    def __init__(self, window):
        wx.FileDropTarget.__init__(self)
        self.window = window

    def OnDropFiles(self, x, y, filenames):

        for name en filenames:
            try:
                file = open(name, 'r')
                text = file.read()
                self.window.WriteText(text)
                file.close()
            except IOError, error:
                dlg = wx.MessageDialog(None, 'Error opening file\n' + 
str(error))
                dlg.ShowModal()
            except UnicodeDecodeError, error:
                dlg = wx.MessageDialog(None, 'Cannot open non ascii files\n' 
+ str(error))
                dlg.ShowModal()

class DropFile(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size = (450, 400))

        self.text = wx.TextCtrl(self, -1, style = wx.TE_MULTILINE)
        dt = FileDrop(self.text)
        self.text.SetDropTarget(dt)
        self.Centre()
        self.Show(True)


app = wx.App()
DropFile(None, -1, 'filedrop.py')
app.MainLoop()
```

Este capítulo describió brevemente arrastrar y soltar operaciones en wxPython.

Internacionalización

En computación, Internacionalización y localización son medios de adaptar software de computadoras para medios ambientes no nativos, especialmente otras naciones y culturas. Internacionalización es el proceso de garantizar que una aplicación es capaz de adaptarse a requerimientos locales, por ejemplo garantizar que el sistema de escritura local puede ser mostrado. Localización es el proceso de adaptar el software para sea tan familiar como posible para una ubicación específica, para mostrar texto en el lenguaje local y usando convenciones locales para mostrar cosas tales como unidades de medida. (wikipedia)

Unicode

Estos son dos construcciones de wxPython. La construcción ansi y la construcción unicode. Si buscamos crear aplicaciones y usamos wxPython en lenguajes otros que el inglés, debemos tener la construcción unicode.

Unicode es un estándar de la industria que permite a las computadoras representar consistentemente y manipular texto expresado en cualquier sistema de escritura del mundo. Esto es una codificación de caracteres estándar la cual usa 16 bits para almacenar caracteres. La codificación **ASCII** traditional usa solamente 8 bits.

Primero, necesitamos obtener el codificación unicode de las palabras Лев Николаевич Толстой Анна Каренина.

```
>>> unicode(u'Лев Николаевич Толстой Анна Каренина')
u'\u041b\u0435\u0432 \u041d\u0438\u043aa\u0430\u0301\u0435\u0432\u0438\u0447
\u0422\u043e\u043b\u0441o\u0439 \u0410\u043d\u043d\u0430
\u041a\u0430\u0440\u0435\u043d\u0438\u043d\u0430'
```

Lanzamos la terminal python y usamos la función llamada *unicode()*. Aviso, en el ejemplo, usamos caracteres \n\ adicionales para dividir palabras dentro de dos líneas.

```
#!/usr/bin/python

import wx

text = u'\u041b\u0435\u0432 \u041d\u0438\u043a\u043e\u043b\u0430\
\u0435\u0432\u0438\u0447 \u0422\u043e\u043b\u0441\u0442\u043e\u0439 \n\
\u0410\u043d\u043d\u0430 \u041a\u0430\u0440\u0435\u043d\u0438\u043d\u0430'


class Unicode(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(250, 150))

        self.Bind(wx.EVT_PAINT, self.OnPaint)

        self.Centre()
        self.Show(True)

    def OnPaint(self, event):
        dc = wx.PaintDC(self)
        dc.DrawText(text, 50, 50)
```

```
app = wx.App()
Unicode(None, -1, 'Unicode')
app.MainLoop()
```

En el ejemplo, dibujamos Anna Karenina en ruso azbuka[1] sobre la ventana.

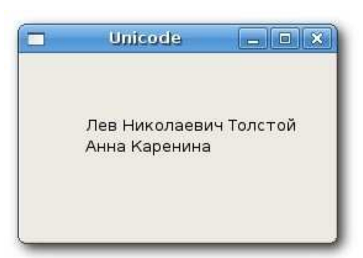


Figura: Unicode

Locale (lugar)

Un lugar es un objeto que define el lenguaje del usuario, país, número de formato, formato de carta, formato de moneda etc. Una variante local tiene el siguiente parametro.

```
[lenguaje[_territory][.codeset][@modifier]]
```

Por ejemplo, **de_AT.utf8** es un local alemán usado en Austria, con codeset UTF8.

```
#!/usr/bin/python
# locale.py

import wx
import time
import locale

class Locale(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(250, 420))

        panel = wx.Panel(self, -1)


        tm = time.localtime()

        font = wx.Font(10, wx.DEFAULT, wx.NORMAL, wx.BOLD)
        us = wx.StaticText(self, -1, 'United States', (25, 20))
        us.SetFont(font)

        wx.StaticLine(self, -1, (25, 50), (200 ,1))

        locale.setlocale(locale.LC_ALL, '')
        date = time.strftime('%x', tm)
        time_ = time.strftime('%X', tm)
        curr =  locale.currency(100000)

        wx.StaticText(self, -1, 'date: ', (25, 70))
        wx.StaticText(self, -1, 'time: ', (25, 90))
```

1 Nota del Traductor: Alfabeto Cirílico Serbio

```
        wx.StaticText(self, -1, 'currency: ', (25, 110))

        wx.StaticText(self, -1, str(date), (125, 70))
        wx.StaticText(self, -1, str(time_), (125, 90))
        wx.StaticText(self, -1, str(curr), (125, 110))

        de = wx.StaticText(self, -1, 'Germany', (25, 150))
        de.SetFont(font)

        wx.StaticLine(self, -1, (25, 180), (200,1))

        locale.setlocale(locale.LC_ALL, ('de_DE', 'UTF8'))
        date = time.strftime('%x', tm)
        time_ = time.strftime('%X', tm)
        curr =  locale.currency(100000)

        wx.StaticText(self, -1, 'date: ', (25, 200))
        wx.StaticText(self, -1, 'time: ', (25, 220))
        wx.StaticText(self, -1, 'currency: ', (25, 240))
        wx.StaticText(self, -1, date, (125, 200))
        wx.StaticText(self, -1, time_, (125, 220))
        wx.StaticText(self, -1, curr, (125, 240))

        de = wx.StaticText(self, -1, 'Slovakia', (25, 280))
        de.SetFont(font)

        wx.StaticLine(self, -1, (25, 310), (200,1))

        locale.setlocale(locale.LC_ALL, ('sk_SK', 'UTF8'))
        date = time.strftime('%x', tm)
        time_ = time.strftime('%X', tm)
        curr =  locale.currency(100000)

        wx.StaticText(self, -1, 'date: ', (25, 330))
        wx.StaticText(self, -1, 'time: ', (25, 350))
        wx.StaticText(self, -1, 'currency: ', (25, 370))

        wx.StaticText(self, -1, str(date), (125, 330))
        wx.StaticText(self, -1, str(time_), (125, 350))
        wx.StaticText(self, -1, str(curr), (125, 370))

        self.Centre()
        self.Show(True)

app = wx.App()
Locale(None, -1, 'Locale')
app.MainLoop()
```

Usamos el estándar incorporado el módulo **locale** para trabajar con configuraciones locales. En nuestro ejemplo, mostraremos varios formatos de fecha, tiempo y moneda en el EEUU, Alemania y Eslovaquia.

```
locale.setlocale(locale.LC_ALL, ('de_DE', 'UTF8'))
```

Aquí fijamos un objeto local para Alemania. **LC_ALL** es una combinación de varias configuraciones

locales, p.e. `LC_TIME`, `LC_MONETARY`, `LC_NUMERIC`.

```
date = time.strftime('%x', tm)
time_ = time.strftime('%X', tm)
curr =  locale.currency(100000)
```

Esta función llama al objeto local corriente.

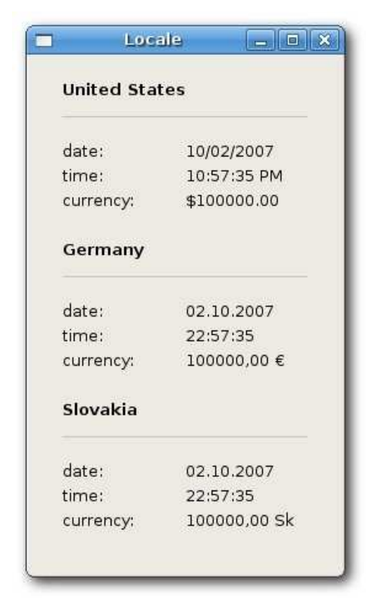


Figura: Locale

Hora Mundial

En un momento específico, tenemos diferentes horarios en paises a través del mundo. Nuestro globo está dividido dentro de husos horarios. No es poco común para los programadores hacer frente con tales tareas. wxPython viene con un objeto *wx.DateTime*. De acuerdo a la documentación, la clase wxDateTime representa un momento absoluto en el tiempo.

```
#!/usr/bin/python

import wx
import time

class WorldTime(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(270, 280))

        self.panel = wx.Panel(self, -1)
        self.panel.SetBackgroundColour('#000000')
        font = wx.Font(12, wx.FONTFAMILY_DEFAULT,
                wx.FONTSTYLE_NORMAL, wx.FONTWEIGHT_BOLD, False, 'Georgia')

        self.dt = wx.DateTime()

        self.tokyo = wx.StaticText(self.panel, -1,
                self.dt.FormatTime() , (20, 20))
        self.tokyo.SetForegroundColour('#23f002')
        self.tokyo.SetFont(font)

        self.moscow = wx.StaticText(self.panel, -1,
```

```
                self.dt.FormatTime() , (20, 70))
        self.moscow.SetForegroundColour('#23f002')
        self.moscow.SetFont(font)

        self.budapest = wx.StaticText(self.panel, -1,
                self.dt.FormatTime() , (20, 120))
        self.budapest.SetForegroundColour('#23f002')
        self.budapest.SetFont(font)

        self.london = wx.StaticText(self.panel, -1,
                self.dt.FormatTime() , (20, 170))
        self.london.SetForegroundColour('#23f002')
        self.london.SetFont(font)

        self.newyork = wx.StaticText(self.panel, -1,
                self.dt.FormatTime() , (20, 220))
        self.newyork.SetForegroundColour('#23f002')
        self.newyork.SetFont(font)

        self.OnTimer(None)

        self.timer = wx.Timer(self)
        self.timer.Start(1000)
        self.Bind(wx.EVT_TIMER, self.OnTimer)

        self.Centre()
        self.Show(True)

    def OnTimer(self, evt):
        now = self.dt.Now()
        self.tokyo.SetLabel('Tokyo: ' + str(now.Format(('%a %T'),
                wx.DateTime.GMT_9)))
        self.moscow.SetLabel('Moscow: ' + str(now.Format(('%a %T'),
                wx.DateTime.MSD)))
        self.budapest.SetLabel('Budapest: ' +  str(now.Format(('%a %T'),
                wx.DateTime.CEST)))
        self.london.SetLabel('London: ' + str(now.Format(('%a %T'),
                wx.DateTime.WEST)))
        self.newyork.SetLabel('New York: ' + str(now.Format(('%a %T'),
                wx.DateTime.EDT)))


app = wx.App()
WorldTime(None, -1, 'World Time')
app.MainLoop()
```

En el código ejemplo, Mostraremos tiempo actual en Tokio, Moscú, Budapest, Londres y New York.

```
self.dt = wx.DateTime()
```

Aquí creamos un objeto *wx.DateTime*.

```
now = self.dt.Now()
```

tomamos el "momento absoluto" en el tiempo.

```
self.tokyo.SetLabel('Tokyo: ' + str(now.Format(('%a %T'),
    wx.DateTime.GMT_9)))
```

Esta linea de código fija el tiempo al formato apropiado. El especificador de conversión %a es un nombre del día de la semana abreviado de acuerdo al localización actual. El %T es la hora del día usando números decimales usando el formato %H:%M:%S. El segundo parámetro del método Format() especifica el zona horaria. GMT_9 es usado para Japon, EDT (Eastern Daylight Saving Time) es usado en New York etc.

El código ejemplo fue verificado con el sitio web timeanddate.com.



Figura: Hora Mundial

Clasificación

Las configuraciones locales también afectan la forma, como las cadenas están siendo clasificadas. Por ejemplo lenguaje Hungaro tiene algunos caracteres que no existen en lenguaje Eslovaco o lenguaje Inglés. Algunos lenguajes tienen acentos, otros no.

```
#!/usr/bin/python
# collate.py

import wx
import locale

ID_SORT = 1

words = [u'Sund', u'S\xe4bel', u'S\xfcnde', u'Schl\xe4fe', u'Sabotage']


class Collate(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(300, 220))

        panel = wx.Panel(self, -1)
        hbox = wx.BoxSizer(wx.HORIZONTAL)

        self.listbox = wx.ListBox(panel, -1)
        for i in words:
            self.listbox.Append(i)
        hbox.Add(self.listbox, 1, wx.EXPAND | wx.ALL, 20)
```

```
        btnPanel = wx.Panel(panel, -1)
        vbox = wx.BoxSizer(wx.VERTICAL)
        new = wx.button(btnPanel, ID_SORT, 'Sort', size=(90, 30))

        self.Bind(wx.EVT_BUTTON, self.OnSort, id=ID_SORT)

        vbox.Add((-1, 20))
        vbox.Add(new)

        btnPanel.SetSizer(vbox)
        hbox.Add(btnPanel, 0.6, wx.EXPAND | wx.RIGHT, 20)
        panel.SetSizer(hbox)

        locale.setlocale(locale.LC_COLLATE, ('de_DE', 'UTF8'))

        self.Centre()
        self.Show(True)

    def OnSort(self, event):
        self.listbox.Clear()
        words.sort( lambda a,b: locale.strcoll(a, b) )
        for i in words:
            self.listbox.Append(i)


app = wx.App()
Collate(None, -1, 'Collate')
app.MainLoop()
```

En nuestro ejemplo, tomamos 5 palabras alemanas desde el diccionario. Por omisión, la función *sort()* clasifica estas palabras de esta manera: Sabotage, Schläfe, Sund, Säbel, Sünde. Esto es incorrecto, porque en alfabeto aleman el caracter **ä** precede al caracter **a**. Para obtener la clasificación correcta, debemos usar funciones locale.

```
locale.setlocale(locale.LC_COLLATE, ('de_DE', 'UTF8'))
```

Aquí fijamos el colección alemana. Podríamos usar la opción *LC_ALL* o el más específico *LC_COLLATE*.

```
words.sort( lambda a,b: locale.strcoll(a, b) )
```

El truco es usar una nueva función para comparar dentro de la función *sort()*. Definimos una función lambda anónima. La función *strcoll()* compara dos cadenas y retorna -1, 0, 1 exactamente como por omisión, pero se necesitan las configuraciones locales (la colección, collate) dentro de la cuenta. De esta manera tenemos la clasificación correcta de las palabras.



Figura: Collate

Traducción Simple

En el siguiente ejemplo, demostraremos una traducción muy básica.

Un programador tiene dos opciones. O usar el GNU gettext o usar los catálogos wxPython. Ambos sistemas son compatibles.

wxPython tiene una clase *wx.Locale*, la cual es una base para el uso de catálogos de mensajes. Cada traducción tiene un catálogo. Por ejemplo, buscamos traducir una cadena dentro del lenguaje aleman. Primero, debemos asegurar, que tenemos soporte para el lenguaje aleman.

```
$ locale -a
C
de_AT.utf8
de_BE.utf8
de_CH.utf8
de_DE.utf8
de_LU.utf8
en_AU.utf8
en_BW.utf8
en_CA.utf8
en_DK.utf8
en_GB.utf8
en_HK.utf8
en_IE.utf8
en_IN
en_NZ.utf8
en_PH.utf8
en_SG.utf8
en_US.utf8
en_ZA.utf8
en_ZW.utf8
POSIX
sk_SK.utf8
```

Para verificar qué lenguajes son soportados, usamos el comando *locale*. Sobre mi sistema, Tengo soporte para los lenguajes inglés, aleman y eslovaco. El lenguaje inglés y el lenguaje aleman tienen diferentes dialectos, es por eso que tenemos muchas opciones. Fijese la cadena **utf8**. De esta manera, que el sistema usa codificación utf8 para trabajar con cadenas.

A continuación escribimos nuestro código de ejemplo. Ponemos la cadena que está para ser traducida dentro de this _(), o podemos usar la llamada *wx.GetTranslation()*.

```
#!/usr/bin/python
```

```
import wx

class Translation(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(220, 100))

        panel = wx.Panel(self, -1)

        mylocale = wx.Locale()
        mylocale.AddCatalogLookupPathPrefix('.')
        mylocale.AddCatalog('simple_de')

        _ = wx.GetTranslation

        wx.StaticText(panel, -1, _("hello"), (10, 10))
        #wx.StaticText(panel, -1, wx.GetTranslation('hello'), (10, 10))

        self.Centre()
        self.Show(True)

app = wx.App()
Translation(None, -1, 'Translation')
app.MainLoop()
```

A continuación creamos un archivo llamado PO. Esto es un simple archivo de texto, el cual es el traductor que usamos para traducir las cadenas.

```
pygettext -o simple_de.po simple.py
```

Para crear un archivo po, usamos el comando **pygettext**. Para entender completamente el formato del archivo po, consulte el manual de gnu gettext.

```
"Content-Type: text/plain; charset=utf-8\n"
```

Editamos el archivo simple_de.po. Debemos especificar el conjunto de caracteres. En nuestro caso es utf-8.

```
#: simple.py:17
msgid "hello"
msgstr "Grüß Gott"
```

Aquí proveemos una traducción para la cadena hello.

La última cosa que hacemos es crear un catálogo de mensaje binario.

```
msgfmt --output-file simple_de.mo simple_de.po
```

Para producir un archivo mo, llamamos el comando **msgfmt**.



Figura: Simple traducción

En este capítulo, hemos trabajado con caracteres unicode.

Trabajar con bases de datos

Aplicaciones usando base de datos son una gran parte de todas las aplicaciones alguna vez desarrolladas. Y aquellas también serán definitivamente en el futuro. La mayoría de ellas son aplicaciones de negocios. Las compañias trabajan con grandes cantidades de datos y aquellas necesitan software para esto.

Algunos kits de herramientas GUI son orientadas al desarrollo de aplicaciones de negocios. Por ejemplo WinForms o Swing. Estos proveen componentes que son adaptados al desarrollo de aplicaciones de negocios. Un componente grilla de datos es un buen ejemplo.

La base de datos es una colección estructurada de datos que es almacenada en una computadora. Un programa de computadora, que gestiona y consulta una base de datos es llamado un **Database Management System (DBMS),** o SGBD. Hace unos treinta años, los DBMS solo estaban disponibles en laboratorios de investigación de grandes compañias tales como IBM. Después, aquellos comenzaron a expandirse. Pero aquellos eran muy caros. En estos días, podemos encontrados DBMS en todas partes. Sobre la Web, sobre nuestras computadoras personales, en varios dispositivos móviles o portables. Podemos tener muchas diferentes bases de datos por poco o ningún dinero que podrían costar miles de dólares en el pasado.

Estos sobre varios modelos de base de datos. El most significativos modelo de base de datos es el **modelo de base de datos relacional (RDBMS)**. Los datos son dividido dentro de tablas. Entre estas tablas definimos relaciones. Estos son varios bien conocidos DBMS comerciales también tanto como de fuentes abiertas (open source).

Comercial RDBMS	Open Source RDBMS
Oracle	MySQL
Sybase	PostgreSQL
MS SQL	Firebird
Access	SQLite

El lenguaje de programación Python tiene módulos para todos los RDBMS de arriba.

SQLite

A partir de las series Python 2.5.x, una biblioteca SQLite es incluida en el lenguaje Python. SQLite es una biblioteca pequeña e integrada. De esta manera los programadores pueden integrar la biblioteca en el interior de su aplicaciones. No es necesario servidor para trabajar con SQLite. Por lo tanto SQLite es también llamada un motor de base de datos SQL con cero configuración.

SQLite tiene las siguientes características:

- trabaja con transacciones
- no necesita administración
- pequeña cantidad de código, menos que 250 KB
- simple para usar y rápida
- estructura de archivo de base de datos simple
- soportes bases de datos de hasta 2 tebibytes (2^{41} bytes) en tamaño

SQLite soporta estos tipo de datos:

- TEXT
- INTEGER
- FLOAT
- BLOB
- NULL

Antes que comencemos a trabajar con SQLite, definimos algunos términos importantes. Una **query** de base de datos es una búsqueda de información desde una base de datos. Una consulta esta escrita en lenguaje SQL. **Structured Query lenguaje** (SQL) es un lenguaje de computadora usado para crear, recuperar, actualizar y eliminar datos desde la base de datos. Éste fue desarrollado por la corporación IBM. El lenguaje SQL tiene tres sub conjuntos.

- DML
- DDL
- DCL

El DML (Lenguaje de Manipulación de Datos) es usado para agregar, actualizar y eliminar datos. SQLite entiende los comandos sql **insert**, **update** y **delete**. El DDL (Lenguaje de Definición de Datos) es usado definir tablas nuevas y registros. SQLite tiene los comandos sql **create**, **drop**, **alter** de este grupo. El DCL (Lenguaje de Control de Datos) es usado para set privilegios para los usuarios base de datos. SQLite no tiene este subconjunto.

Un cursor es un objeto de una base de datos usado para atravesar los resultados de una consultas SQL. Una transacción es una unidad de operación con un sistema de gestión de base de datos. Ésta puede contener una o más consultas. Las transacciones son usadas to asegurar la integridad de datos en una base de datos. Si todo es ok. Las transacciones son comprometidas. Si una o más consultas fallan. Las transacciones son deshechas. La base de datos que soporta transacciones son llamadas bases de datos transaccionales. Una base de datos SQLite es una base de datos transaccional. Un **conjunto de resultados SQL** es un conjunto de filas y metadatos acerca de la consulta desde una base de datos. Esto es, un conjunto de registros que resultan de la corrida de una consulta. Una simple unidad de datos estructurados dentro de una tabla de base de datos es llamada un registro o fila.

sqlite3

La biblioteca SQLite incluye una pequeña utilidad de línea de comando llamada sqlite3. Esto es usado para entrar manualmente y ejecutar comandos SQL contra una base de datos SQLite. Para lanzar este utilitario, tipeamos sqlite3 dentro de la consola de comandos. El comando está seguido por el nombre de una base de datos. Si la base de datos no existe, una nueva es creada. Trabajamos con sqlite3 con un conjunto definido de punto(dot)comandos. Para mostrar todos los comandos disponibles, tipeamos *.help*. Algunos del comandos son mostrados en la siguiente tabla.

Command	Descripción
.databases	muestra el nombre de una base de datos
.dump table	Vuelca una tabla dentro de un texto de formato SQL
.exit	sale del programa sqlite3
.headers ON|OFF	muestra u ocultar cabeceras de columnas
.help	muestra la ayuda
.mode mode table	cambia modo para un table
.quit	Lo mismo que .exit
.read filename	ejecutar comandos SQL en un archivo
.show	muestra las configuraciones sqlite3
.tables pattern	lista tablas que coinciden con el patrón
.width num num ...	fija width for columnas

Primero, creamos una nueva base de datos llamada people.

```
$ sqlite3 people
SQLite version 3.3.13
Enter ".help" for instructions
sqlite>
```

```
sqlite> .databases
seq  name             file
---  ---------------
-----------------------------------------------------
0    main             /home/vronskij/tmp/people
sqlite> .exit
$
```

Todos los comandos de *sqlite3* comienzan con el caracter punto ".". Para mostrar todos los comandos disponibles, simplemente tipeamos *.help*. El comando*.databases* muestra nuestra base de datos corriente (activa). El comando*.exit* quita el utilitario sqlite3 y retorna a la consola de comandos.

A continuación creamos una tabla.

```
sqlite> .tables
sqlite> create table neighbours(name text, age numeric, remark text);
sqlite> .tables
neighbours
```

El comando*.tables* muestra todos las tablas disponibles en la base de datos. Creamos una tabla llamada neighbours. Nuestra tabla tendrá tres columnas. Podríamos usar tipo de datos texto y numérico. Tenga en cuenta que cada comando SQL es seguido por un punto y coma ";".

Ahora es tiempo de insertar algunos datos reales.

```
sqlite> insert into neighbours values('sandy', 7, 'stubborn');
sqlite> insert into neighbours values('jane', 18, 'beautiful');
sqlite> insert into neighbours values('mark', 28, 'lazy');
sqlite> insert into neighbours values('steven', 34, 'friendly');
sqlite> insert into neighbours values('alice', 17, 'slick');
sqlite> insert into neighbours values('tom', 25, 'clever');
sqlite> insert into neighbours values('jack', 89, 'wise');
sqlite> insert into neighbours values('lucy', 18, 'cute');
```

El comando SQL *select* es probablemente el comando más ampliamente usado del DML (Lenguaje de manipulación de datos).

```
sqlite> select * from neighbours;
sandy|7|stubborn
jane|18|beautiful
mark|28|lazy
steven|34|friendly
alice|17|slick
tom|25|clever
jack|89|wise
lucy|18|cute
```

El sqlite3 tiene varios modos de mostrar datos. Es decir,:

Modo	Descripción
csv	valores separados por comas
column	columnas alineadas a la izquierda
html	código de tabla html
insert	SQL sentencias insert para una tabla
line	un valor por línea

list	valores delimitados por la cadena .separator
tabs	valores separados por tabs

El modo por omisión es el modo lista. Podemos ver las configuraciones corrientes si tipeamos el comando.show.

```
sqlite> .show
     echo: off
  explain: off
  headers: off
     mode: list
nullvalue: ""
   output: stdout
separator: "|"
    width:
```

Prefiero el modo columna. En el siguiente paso cambiamos un poquito las configuraciones por omisión.

```
sqlite> .mode column
sqlite> .headers on
sqlite> .width 10 4 15
sqlite> select * from neighbours;
name        age   remark
----------  ----  ---------------
sandy       7     stubborn
jane        18    beautiful
mark        28    lazy
steven      34    friendly
alice       17    slick
tom         25    clever
jack        89    wise
lucy        18    cute
```

Cambiamos el modo con el comando *.mode* al modo columna. Fijamos cabeceras sobre con el comando*.headers*. Finalmente cambiamos el ancho de cada columna con el comando *.width*. El valor por omisión es diez caracteres.

La copia de seguridad de los datos es el lo mas importante. sqlite3 tiene una solución simple. Utilizamos el comando *.dump*.

```
sqlite> .tables
neighbours
sqlite> .dump neighbours
BEGIN TRANSACTION;
CREATE TABLE neighbours(name text, age numeric, remark text);
INSERT INTO "neighbours" VALUES('sandy',7,'stubborn');
INSERT INTO "neighbours" VALUES('jane',18,'beautiful');
INSERT INTO "neighbours" VALUES('mark',28,'lazy');
INSERT INTO "neighbours" VALUES('steven',34,'friendly');
INSERT INTO "neighbours" VALUES('alice',17,'slick');
INSERT INTO "neighbours" VALUES('tom',25,'clever');
INSERT INTO "neighbours" VALUES('jack',89,'wise');
INSERT INTO "neighbours" VALUES('lucy',18,'cute');
COMMIT;
```

El comando *.dump* transforma la tabla dentro de un formato de texto de un conjunto comandos de

SQL. Estos comandos SQL podrá recrear la tabla al estado original. Copiamos y pegamos estos comandos SQL dentro del archivo de texto neighbours.sql.

En los siguientes pasos eliminamos una tabla y la recreamos desde nuestro archivo.

```
sqlite> drop table neighbours;
sqlite> .tables
sqlite> .read ../neighbours.sql
sqlite> .tables
neighbours
sqlite> select * from neighbours;
name        age         remark
----------  ----------  ----------
sandy       7           stubborn
jane        18          beautiful
mark        28          lazy
steven      34          friendly
alice       17          slick
tom         25          clever
jack        89          wise
lucy        18          cute
```

Eliminamos la tabla neighbours con el comando SQL drop table. El comando *.tables* no muestra la tabla. Entonces tipeamos comando sqlite *.read* para ejecutar todos los comandos SQL en el archivo especificado. Finalmente, we verify nuestras datos.

SQLite Python API

pysqlite es una interfaz python a la biblioteca SQLite. Desde las versiones python 2.5x, está incluida en el lenguaje python. El módulo pysqlite es incluido bajo el nombre de paquete sqlite3.

```
import sqlite3 as lite
```

Pasos simples

- crear un objeto conexión
- crear un objeto cursor
- ejecutar la consulta
- recuperamos los datos (opcional)
- cerramos el cursor y los objetos conexión

para crear una conexión, llamamos el método *connect()* del módulo.

```
import sqlite3 as lite

con = lite.connect('databasename')
con = lite.connect(':memory:')
```

Estos son dos caminos para crear a objeto conexión. Podemos crear una conexión a una base de datos sobre el sistema de archivos. Simplemente especificamos la ruta al nombre de archivo. Podemos también crear una base de datos en memoria. Éste es hecho con una cadena especial ':memory:'.

Lanzamos un interprete Python. Podríamos probar nuestro ejemplos aquí.

```
$ python
Python 2.5.1c1 (release25-maint, Apr  6 2007, 22:02:36)
[GCC 4.1.2 (Ubuntu 4.1.2-0ubuntu4)] on linux2
Type "help", "copyright", "credits" o "license" for more información.
```

```
>>>

>>> import sqlite3 as lite
>>> con = lite.connect('people')
>>> cur = con.cursor()
>>> cur.execute('select name from neighbours')
>>> print cur.fetchall()
[(u'sandy',), (u'jane',), (u'mark',), (u'steven',), (u'alice',), (u'tom',),
(u'jack',), (u'lucy',)]
>>> cur.close()
>>> con.close()
```

Primero importamos el módulo sqlite3. Entonces conectamos a nuestras base de datos people. El archivo base de datos está en nuestro directorio actual. Para crear un objeto cursor, llamamos el método *cursor()* del objeto conexión. Después de que llamamos dos métodos objeto cursor. El método *execute()* ejecuta comandos SQL. El método *fetchall()* recupera todos los datos que tenemos seleccionados. El camino correcto para terminar nuestro trabajo es cerrar el cursor y el objeto conexión.

Confirmando los cambios

La biblioteca SQLite trabaja con transacciones. Ésto es importante para comprender como trabaja esto. De acuerdo a la documentación, para cada sentencia DML, SQLite abre una transacción. Debemos confirmar y aplicar nuestros cambios. Para cada sentencia DCL, la biblioteca SQLite confirma automáticamente los cambios. Demostraremos esto en ejemplos cortos.

```
>>> cur.execute("update neighbours set age=29 donde name='lucy'")
>>> cur.execute("select age from neighbours donde name='lucy'")
>>> print cur.fetchone()
(29,)
>>> cur.close()
>>> con.close()
>>> (CTRL + D)
$ sqlite3 people
sqlite> select age from neighbours donde name='lucy';
18
```

¿Qué salió mal? No confirmamos nuestros cambios. Cuando ejecutamos la sentencia select usando la api sqlite de python, recibimos resultados dentro de un contexto de transacción. Los cambios no fueron escritos realmente en la base de datos. Cuando verificamos los datos en el utilitario sqlite3, tenemos la edad 18. El dato no fue cambiado.

```
>>> cur.execute("update neighbours set age=29 where name='lucy'")
>>> con.commit()
>>> cur.close()
>>> con.close()
>>> (CTRL + D)
$ sqlite3 people
sqlite> select age from neighbours where name='lucy';
29
```

Después de confirmar nuestros cambios con el método *commit()* del objeto conexión, los cambios de los datos son escritos realmente en la base de datos.

En el ejemplo siguiente demostramos que las sentencias DDL son confirmadas automáticamente.

Podríamos usar el comando create table, la cual es una parte del lenguaje DDL.

```
>>> cur.execute('create table relatives(name text, age numeric)')
>>> cur.close()
>>> con.close()
>>> (CTRL + D)
$ sqlite3 people
sqlite> .tables
neighbours  relatives
```

Esta es una cosa más a mencionar. Podemos crear una conexión, la cual podrá automáticamente confirmar todos nuestros cambios. Éste es hecho, cuando fijamos el parámetro *isolation_level* a None.

```
>>> import sqlite3 as lite
>>> con = lite.connect('people', isolation_level=None)
>>> cur = con.cursor()
>>> cur.execute("insert dentro de neighbours values ('rebecca', 16, 'shy')")
>>> cur.close()
>>> con.close()
>>> (CTRL + D)
$ sqlite3 people
sqlite> select * from neighbours donde name='rebecca';
rebecca|16|shy
sqlite>
```

Autoincremento

Las claves primarias autoincrementales es una característica práctica. Insertamos nuevas filas y la clave es incrementada automáticamente en uno. El implementation de la característica deautoincremento puede diferir entre RDMSs. En el ejemplo siguiente mostraremos como esto es hecho en la base de datos SQLite.

```
sqlite> create table books(id integer primary key autoincrement not null,
name text, author text);
sqlite> insert into books (name, author) values ('anna karenina', 'leo
tolstoy');
sqlite> insert into books (name, author) values ('father goriot', 'honore de
balzac');
sqlite> select * from books;
1|anna karenina|leo tolstoy
2|father goriot|honore de balzac
sqlite>
```

La palabra clave*autoincrement* es usada para crear las claves primarias autoincrementales en SQLite.

Consideraciones de Seguridad

Es posible pero inseguro pasar parámetros de esta manera.

```
bookname = 'atlante illustrato di filosofia'
bookauthor = 'ubaldo nicola'
cur.execute("insert dentro de books(name, author) values ('%s', '%s')" %
(bookname, bookauthor))
```

Éste es vulnerable a ataques. Estos ataques son llamados injección de SQL. No haga esto.

```
>>> import sqlite3 as lite
>>> print lite.paramstyle
qmark
```

La lista de especificaciones de la API de la base de datos para Python tiene los siguientes estilos de paso de parámetros:

- qmark
- numérico
- named
- format
- pyformat

La API de SQLite para Python usa el qmark (question mark) citado. El ejemplo previo reescrito en estilo qmark:

```
 bookname = 'atlante illustrato di filosofia'
 bookauthor = 'ubaldo nicola'
 cur.execute('insert dentro de books(name, author) values (?, ?)',
(bookname, bookauthor))
```

Poniendo todo junto

Hasta el momento tenemos vista a la biblioteca SQLite3, las bases de datos y el lenguaje SQL. Ahora es el momento para ponerlos todo junto con wxPython en un guión simple funcional. El siguiente guión simple podrá hacer solamente una cosa específica. Insertar datos dentro de una tabla. Podríamos usar de nuestra base de datos personas, al tabla neigbours.

```
#!/usr/bin/python
# insertdata.py

import wx
import sqlite3 as lite

class InsertData(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(280, 200))

        panel = wx.Panel(self, -1)

        gs = wx.FlexGridSizer(3, 2, 9, 9)
        vbox = wx.BoxSizer(wx.VERTICAL)
        hbox = wx.BoxSizer(wx.HORIZONTAL)

        name = wx.StaticText(panel, -1, 'Name')
        remark = wx.StaticText(panel, -1, 'Remark')
        age = wx.StaticText(panel, -1, 'Age')
        self.sp = wx.SpinCtrl(panel, -1, '', size=(60, -1), min=1, max=125)
        self.tc1 = wx.TextCtrl(panel, -1, size=(150, -1))
        self.tc2 = wx.TextCtrl(panel, -1, size=(150, -1))

        gs.AddMany([(name), (self.tc1, 1, wx.LEFT, 10),
                (remark), (self.tc2, 1, wx.LEFT, 10),
                (age), (self.sp, 0, wx.LEFT, 10)])

        vbox.Add(gs, 0, wx.ALL, 10)
```

```
        vbox.Add((-1, 30))

        insert = wx.button(panel, -1, 'Insert', size=(-1, 30))
        cancel = wx.button(panel, -1, 'Cancel', size=(-1, 30))
        hbox.Add(insert)
        hbox.Add(cancel, 0, wx.LEFT, 5)
        vbox.Add(hbox, 0, wx.ALIGN_CENTER | wx.BOTTOM, 10)

        self.Bind(wx.EVT_BUTTON, self.OnInsert, id=insert.GetId())
        self.Bind(wx.EVT_BUTTON, self.OnCancel, id=cancel.GetId())

        panel.SetSizer(vbox)

        self.Centre()
        self.Show(True)

    def OnInsert(self, event):
        try:
            con = lite.connect('people')
            cur = con.cursor()
            name = self.tc1.GetValue()
            age = self.sp.GetValue()
            remark = self.tc2.GetValue()
            cur.execute('insert dentro de neighbours values(?, ?, ?)', 
(name, age, remark))
            con.commit()
            cur.close()
            con.close()

        except lite.Error, error:
            dlg = wx.MessageDialog(self, str(error), 'Error occured')
            dlg.ShowModal()

    def OnCancel(self, event):
        self.Close()

app = wx.App()
InsertData(None, -1, 'Insert Dialog')
app.MainLoop()
```

```
 gs = wx.FlexGridSizer(3, 2, 9, 9)
```

En nuestro cuadro de diálogo usamos items de diferentes tamaños. Es por eso que tenemos elegido el *wx.FlexGridSizer. Los items* en *wx.GridSizer* tienen siempre el mismo tamaño.

```
 name = self.tc1.GetValue()
 age = self.sp.GetValue()
 remark = self.tc2.GetValue()
 cur.execute('insert dentro de neighbours values(?, ?, ?)', (name, age, 
remark))
```

Ésta es la parte crucial del código. En las tres primeras líneas, tomamos los valores que el usuario tiene insertados. Estos valores son insertados dentro de la base de datos con el código SQL apropiado.

```
 except lite.Error, error:
     dlg = wx.MessageDialog(self, str(error), 'Error occured')
```

```
dlg.ShowModal()
```

Tenemos colocada nuestro codigo relacionado a la base de datos entre la clausula try - catch. Ésto es porque trabajar con datos y bases de datos es propenzo a errores. La excepción *Error* es una clase base para todas las otras excepciones implementadas en la biblioteca SQLite.



Figura: insertdata.py diálogo

En esta parte del tutorial de wxPython, trabajamos con la base de datos SQLite.

Esqueletos de aplicaciones en wxPython

En esta sección, vamos a crear algunos esqueletos de aplicaciones. Nuestros guiones podrán trabajar fuera de la interfaz pero no podran implementar la funcionalidad. El objetivo es mostrar, como varias interfaces GUI bien conocidas se podrían hacer en wxPython.

Administrador de Archivos

File Hunter es un esqueleto de un administrador de archivos. Éste copia el mirador del Krusader, el mejor administrador de archivos disponibles sobre sistemas Unix. Si usted hace doble clic sobre el componente splitter, éste podrá dividir el File Hunter dentro de dos partes con el mismo ancho. Lo mismo pasa, si usted redimensiona la ventana principal.



Figura: Filemanager.py

```
#!/usr/bin/python

import wx
import os
import time


ID_botón=100
ID_EXIT=200
ID_SPLITTER=300

class MyListCtrl(wx.ListCtrl):
    def __init__(self, parent, id):
        wx.ListCtrl.__init__(self, parent, id, style=wx.LC_REPORT)

        files = os.listdir('.')
        images = ['images/empty.png', 'images/folder.png',
'images/source_py.png',
                'images/image.png', 'images/pdf.png', 'images/up16.png']

        self.InsertColumn(0, 'Name')
        self.InsertColumn(1, 'Ext')
        self.InsertColumn(2, 'Size', wx.LIST_FORMAT_RIGHT)
        self.InsertColumn(3, 'Modified')

        self.SetColumnWidth(0, 220)
        self.SetColumnWidth(1, 70)
```

```
        self.SetColumnWidth(2, 100)
        self.SetColumnWidth(3, 420)

        self.il = wx.ImageList(16, 16)
        for i in images:
            self.il.Add(wx.Bitmap(i))
        self.SetImageList(self.il, wx.IMAGE_LIST_SMALL)

        j = 1
        self.InsertStringItem(0, '..')
        self.SetItemImage(0, 5)

        for i in files:
            (name, ext) = os.path.splitext(i)
            ex = ext[1:]
            size = os.path.getsize(i)
            sec = os.path.getmtime(i)
            self.InsertStringItem(j, name)
            self.SetStringItem(j, 1, ex)
            self.SetStringItem(j, 2, str(size) + ' B')
            self.SetStringItem(j, 3, time.strftime('%Y-%m-%d %H:%M',
                time.localtime(sec)))

            if os.path.isdir(i):
                self.SetItemImage(j, 1)
            elif ex == 'py':
                self.SetItemImage(j, 2)
            elif ex == 'jpg':
                self.SetItemImage(j, 3)
            elif ex == 'pdf':
                self.SetItemImage(j, 4)
            else:
                self.SetItemImage(j, 0)

            if (j % 2) == 0:
                self.SetItemBackgroundColour(j, '#e6f1f5')
            j = j + 1


class FileHunter(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, -1, title)

        self.splitter = wx.SplitterWindow(self, ID_SPLITTER,
style=wx.SP_BORDER)
        self.splitter.SetMinimumPaneSize(50)

        p1 = MyListCtrl(self.splitter, -1)
        p2 = MyListCtrl(self.splitter, -1)
        self.splitter.SplitVertically(p1, p2)

        self.Bind(wx.EVT_SIZE, self.OnSize)
        self.Bind(wx.EVT_SPLITTER_DCLICK, self.OnDoubleClick,
id=ID_SPLITTER)

        filemenu= wx.Menu()
```

```
        filemenu.Append(ID_EXIT,"E&xit"," Terminate el program")
        editmenu = wx.Menu()
        netmenu = wx.Menu()
        showmenu = wx.Menu()
        configmenu = wx.Menu()
        helpmenu = wx.Menu()

        menubar = wx.MenuBar()
        menuBar.Append(filemenu,"&File")
        menuBar.Append(editmenu, "&Edit")
        menuBar.Append(netmenu, "&Net")
        menuBar.Append(showmenu, "&Show")
        menuBar.Append(configmenu, "&Config")
        menuBar.Append(helpmenu, "&Help")
        self.SetMenuBar(menuBar)
        self.Bind(wx.EVT_MENU, self.OnExit, id=ID_EXIT)

        tb = self.CreateToolBar( wx.TB_HORIZONTAL | wx.NO_BORDER |
                wx.TB_FLAT | wx.TB_TEXT)
        tb.AddSimpleTool(10, wx.Bitmap('images/previous.png'), 'Previous')
        tb.AddSimpleTool(20, wx.Bitmap('images/up.png'), 'Up uno directory')
        tb.AddSimpleTool(30, wx.Bitmap('images/home.png'), 'Home')
        tb.AddSimpleTool(40, wx.Bitmap('images/refresh.png'), 'Refresh')
        tb.AddSeparator()
        tb.AddSimpleTool(50, wx.Bitmap('images/write.png'), 'Editor')
        tb.AddSimpleTool(60, wx.Bitmap('images/terminal.png'), 'Terminal')
        tb.AddSeparator()
        tb.AddSimpleTool(70, wx.Bitmap('images/help.png'), 'Help')
        tb.Realize()

        self.sizer2 = wx.BoxSizer(wx.HORIZONTAL)

        button1 = wx.button(self, ID_botón + 1, "F3 View")
        button2 = wx.button(self, ID_botón + 2, "F4 Edit")
        button3 = wx.button(self, ID_botón + 3, "F5 Copy")
        button4 = wx.button(self, ID_botón + 4, "F6 Move")
        button5 = wx.button(self, ID_botón + 5, "F7 Mkdir")
        button6 = wx.button(self, ID_botón + 6, "F8 Delete")
        button7 = wx.button(self, ID_botón + 7, "F9 Rename")
        button8 = wx.button(self, ID_EXIT, "F10 Quit")

        self.sizer2.Add(button1, 1, wx.EXPAND)
        self.sizer2.Add(button2, 1, wx.EXPAND)
        self.sizer2.Add(button3, 1, wx.EXPAND)
        self.sizer2.Add(button4, 1, wx.EXPAND)
        self.sizer2.Add(button5, 1, wx.EXPAND)
        self.sizer2.Add(button6, 1, wx.EXPAND)
        self.sizer2.Add(button7, 1, wx.EXPAND)
        self.sizer2.Add(button8, 1, wx.EXPAND)

        self.Bind(wx.EVT_BUTTON, self.OnExit, id=ID_EXIT)

        self.sizer = wx.BoxSizer(wx.VERTICAL)
        self.sizer.Add(self.splitter,1,wx.EXPAND)
        self.sizer.Add(self.sizer2,0,wx.EXPAND)
        self.SetSizer(self.sizer)
```

```
        size = wx.DisplaySize()
        self.SetSize(size)

        self.sb = self.CreateStatusBar()
        self.sb.SetStatusText(os.getcwd())
        self.Center()
        self.Show(True)


    def OnExit(self,e):
        self.Close(True)

    def OnSize(self, event):
        size = self.GetSize()
        self.splitter.SetSashPosition(size.x / 2)
        self.sb.SetStatusText(os.getcwd())
        event.Skip()


    def OnDoubleClick(self, event):
        size =  self.GetSize()
        self.splitter.SetSashPosition(size.x / 2)

app = wx.App(0)
FileHunter(None, -1, 'File Hunter')
app.MainLoop()
```

SpreadSheet

Gnumeric, KSpread y OpenOffice Calc son famosas aplicaciones de hoja de cálculo disponibles sobre Unix. El siguiente ejemplo muestra un esqueleto de una aplicación de hoja de cálculo en wxPython.

Las aplicaciones tienen su propia vida. Ésto también es verdad para guiones educacionales. Después de que he realizado la actualización a wx.Python 2.8.1.1, el ejemplo de hoja de cálculo no trabajaba. La línea siguiente fue el problema

```
 toolbar2.AddControl(wx.StaticText(toolbar2, -1, '  '))
```

Por supuesto, nosotros no podemos agregar un componente a él mismo. Pero la versión previa del kit de herramientas estaba contento con él. Bajo la versión actual no funcionó, señalizar el problema. Esto podría o no trabajar sobre Mac y Windows. Originalmente,Yo quería agregar algo de espacio entre los cuadros combinados. Bajo la nueva versión del kit de herramientas se detuvo para trabajar ya sea se me cayó la línea.

Además de la fijación de este error, Yo también limpié el código un poquito y reemplace los métodos depreciados (*AddSimpleTool()*) de la barra de herramientas con uno nuevo (*AddLabelTool()*).

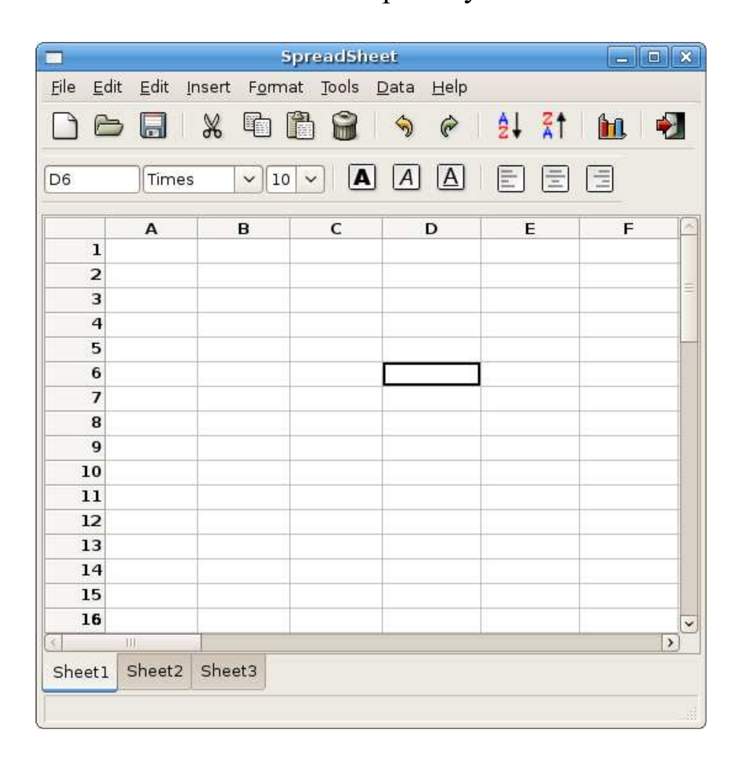


Figura: Spreadsheet

```
#!/usr/bin/python
# spreadsheet.py

from wx.lib import sheet
import wx


class MySheet(sheet.CSheet):
    def __init__(self, parent):
        sheet.CSheet.__init__(self, parent)
        self.row = self.col = 0
        self.SetNumberRows(55)
        self.SetNumberCols(25)

        for i in range(55):
            self.SetRowSize(i, 20)

    def OnGridSelectCell(self, event):
        self.row, self.col = event.GetRow(), event.GetCol()
        control = self.GetParent().GetParent().position
        value =  self.GetColLabelValue(self.col) + 
self.GetRowLabelValue(self.row)
        control.SetValue(value)
        event.Skip()

class Newt(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, -1, title, size = (550, 500))

        fonts = ['Times Nuevos Roman', 'Times', 'Courier', 'Courier New', 
'Helvetica',
                'Sans', 'verdana', 'utkal', 'aakar', 'Arial']
```

```
        font_sizes = ['10', '11', '12', '14', '16']

        box = wx.BoxSizer(wx.VERTICAL)
        menubar = wx.MenuBar()

        menu1 = wx.Menu()
        menuBar.Append(menu1, '&File')
        menu2 = wx.Menu()
        menuBar.Append(menu2, '&Edit')
        menu3 = wx.Menu()
        menuBar.Append(menu3, '&Edit')
        menu4 = wx.Menu()
        menuBar.Append(menu4, '&Insert')
        menu5 = wx.Menu()
        menuBar.Append(menu5, 'F&ormat')
        menu6 = wx.Menu()
        menuBar.Append(menu6, '&Tools')
        menu7 = wx.Menu()
        menuBar.Append(menu7, '&Data')
        menu8 = wx.Menu()
        menuBar.Append(menu8, '&Help')

        self.SetMenuBar(menuBar)

        toolbar1 = wx.ToolBar(self, -1, style= wx.TB_HORIZONTAL)
        toolbar1.AddLabelTool(-1, '', wx.Bitmap('icons/stock_new.png'))
        toolbar1.AddLabelTool(-1, '', wx.Bitmap('icons/stock_open.png'))
        toolbar1.AddLabelTool(-1, '', wx.Bitmap('icons/stock_save.png'))
        toolbar1.AddSeparator()
        toolbar1.AddLabelTool(-1, '', wx.Bitmap('icons/stock_cut.png'))
        toolbar1.AddLabelTool(-1, '', wx.Bitmap('icons/stock_copy.png'))
        toolbar1.AddLabelTool(-1, '', wx.Bitmap('icons/stock_paste.png'))
        toolbar1.AddLabelTool(-1, '',  wx.Bitmap('icons/stock_delete.png'))
        toolbar1.AddSeparator()
        toolbar1.AddLabelTool(-1, '', wx.Bitmap('icons/stock_undo.png'))
        toolbar1.AddLabelTool(-1, '', wx.Bitmap('icons/stock_redo.png'))
        toolbar1.AddSeparator()
        toolbar1.AddLabelTool(-1, '', wx.Bitmap('icons/incr22.png'))
        toolbar1.AddLabelTool(-1, '', wx.Bitmap('icons/decr22.png'))
        toolbar1.AddSeparator()
        toolbar1.AddLabelTool(-1, '', wx.Bitmap('icons/chart.xpm'))
        toolbar1.AddSeparator()
        toolbar1.AddLabelTool(-1, '',  wx.Bitmap('icons/stock_exit.png'))

        toolbar1.Realize()

        toolbar2 = wx.ToolBar(self, wx.TB_HORIZONTAL | wx.TB_TEXT)

        self.position = wx.TextCtrl(toolbar2)
        font = wx.ComboBox(toolbar2, -1, value = 'Times', choices=fonts,
size=(100, -1),
                style=wx.CB_DROPDOWN)
        font_height = wx.ComboBox(toolbar2, -1, value = '10',
choices=font_sizes,
                size=(50, -1), style=wx.CB_DROPDOWN)
```

```
        toolbar2.AddControl(self.position)
        toolbar2.AddControl(font)
        toolbar2.AddControl(font_height)
        toolbar2.AddSeparator()
        bold = wx.Bitmap('icons/stock_text_bold.png')
        toolbar2.AddCheckTool(-1, bold)
        italic = wx.Bitmap('icons/stock_text_italic.png')
        toolbar2.AddCheckTool(-1, italic)
        bajo = wx.Bitmap('icons/stock_text_underline.png')
        toolbar2.AddCheckTool(-1, under)
        toolbar2.AddSeparator()
        toolbar2.AddLabelTool(-1, '',
wx.Bitmap('icons/text_align_left.png'))
        toolbar2.AddLabelTool(-1, '',
wx.Bitmap('icons/text_align_center.png'))
        toolbar2.AddLabelTool(-1, '',
wx.Bitmap('icons/text_align_right.png'))

        box.Add(toolbar1, border=5)
        box.Add((5,5) , 0)
        box.Add(toolbar2)
        box.Add((5,10) , 0)

        toolbar2.Realize()
        self.SetSizer(box)
        notebook = wx.Notebook(self, -1, style=wx.RIGHT)

        sheet1 = MySheet(notebook)
        sheet2 = MySheet(notebook)
        sheet3 = MySheet(notebook)
        sheet1.SetFocus()

        notebook.AddPage(sheet1, 'Sheet1')
        notebook.AddPage(sheet2, 'Sheet2')
        notebook.AddPage(sheet3, 'Sheet3')

        box.Add(notebook, 1, wx.EXPAND)

        self.CreateStatusBar()
        self.Centre()
        self.Show(True)

app = wx.App()
Newt(None, -1, 'SpreadSheet')
app.MainLoop()
```

Mucho del código construcciones de los menús y Barras de Herramientas. Además que, éste es un ejemplo bastante simple.

```
class MySheet(sheet.CSheet):
    def __init__(self, parent):
        sheet.CSheet.__init__(self, parent)
        self.row = self.col = 0
        self.SetNumberRows(55)
        self.SetNumberCols(25)
```

```
        for i in range(55):
            self.SetRowSize(i, 20)
```

La clase MySheet hereda desde la clase CSheet, la cual está localizada en el módulo wx.lib. Éste es basicamente un componente wx.Grid con alguna funcionalidad adicional. Fijamos el tamaño de la fila a 20px. Éste es puramente para fines estéticos.

```
control = self.GetParent().GetParent().position
```

La posición del control de texto muestra la celda seleccionada del componente grilla. Éste es el primer componente de la segunda barra de herramientas. Siendo en el interior de una clase MySheet, necesitamos obtener una referencia al control de texto, la cual es definido en la clase Newt. MySheet es un hijo del notebook. Y notebook es un hijo de Newt. Así logramos obtener la posición del control de texto al llamar el método *GetParent()* dos veces.

```
notebook = wx.Notebook(self, -1, style=wx.RIGHT)
```

Éste es un error. Bajo la versión actual de wxPython (on GTK+), derecha es al fondo y al fondo es derecha.

Browser

Estos días los navegadores de Internet son una de las más importantes aplicaciones en el IT mundo. Imitamos el aspecto de a Firefox en nuestro guión.

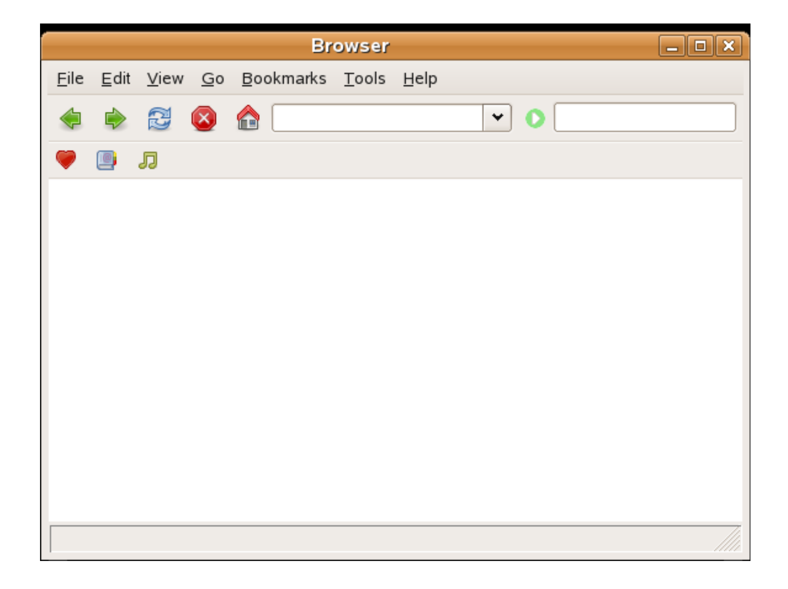


Figura: Browser.py

```
#!/usr/bin/python

import wx
from wx.lib.buttons import GenBitmapTextButton

class Browser(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(450, 400))
        panel = wx.Panel(self, -1)
        panel.SetBackgroundColour('WHITE')

        menubar = wx.MenuBar()
        file = wx.Menu()
        file.Append(1, '&Quit', '')
        edit = wx.Menu()
        view = wx.Menu()
        go = wx.Menu()
        bookmarks = wx.Menu()
        tools = wx.Menu()
        help = wx.Menu()

        menubar.Append(file, '&File')
        menubar.Append(edit, '&Edit')
        menubar.Append(view, '&View')
        menubar.Append(go, '&Go')
        menubar.Append(bookmarks, '&Bookmarks')
        menubar.Append(tools, '&Tools')
        menubar.Append(help, '&Help')

        self.SetMenuBar(menubar)

        vbox = wx.BoxSizer(wx.VERTICAL)
        hbox1 = wx.BoxSizer(wx.HORIZONTAL)
        hbox2 = wx.BoxSizer(wx.HORIZONTAL)
        toolbar1 = wx.Panel(panel, -1, size=(-1, 40))
        back = wx.BitmapButton(toolbar1, -1, wx.Bitmap('icons/back.png'),
                style=wx.NO_BORDER)
```

```
        forward = wx.BitmapButton(toolbar1, -1,
wx.Bitmap('icons/forward.png'),
                style=wx.NO_BORDER)
        refresh = wx.BitmapButton(toolbar1, -1,
wx.Bitmap('icons/refresh.png'),
                style=wx.NO_BORDER)
        stop = wx.BitmapButton(toolbar1, -1, wx.Bitmap('icons/stop.png'),
                style=wx.NO_BORDER)
        home = wx.BitmapButton(toolbar1, -1, wx.Bitmap('icons/home.png'),
                style=wx.NO_BORDER)
        address = wx.ComboBox(toolbar1, -1, size=(50, -1))
        go = wx.BitmapButton(toolbar1, -1, wx.Bitmap('icons/go.png'),
                style=wx.NO_BORDER)
        text = wx.TextCtrl(toolbar1, -1, size=(150, -1))

        hbox1.Add(back)
        hbox1.Add(forward)
        hbox1.Add(refresh)
        hbox1.Add(stop)
        hbox1.Add(home)
        hbox1.Add(address, 1, wx.TOP, 4)
        hbox1.Add(go, 0, wx.TOP | wx.LEFT, 4)
        hbox1.Add(text, 0, wx.TOP | wx.RIGHT, 4)

        vbox.Add(toolbar1, 0, wx.EXPAND)
        line = wx.StaticLine(panel)
        vbox.Add(line, 0, wx.EXPAND)

        toolbar2 = wx.Panel(panel, -1, size=(-1, 30))
        bookmark1 = wx.BitmapButton(toolbar2, -1,
wx.Bitmap('icons/love.png'),
                style=wx.NO_BORDER)
        bookmark2 = wx.BitmapButton(toolbar2, -1,
wx.Bitmap('icons/books.png'),
                style=wx.NO_BORDER)
        bookmark3 = wx.BitmapButton(toolbar2, -1,
wx.Bitmap('icons/sound.png'),
                style=wx.NO_BORDER)
        hbox2.Add(bookmark1, flag=wx.RIGHT, border=5)
        hbox2.Add(bookmark2, flag=wx.RIGHT, border=5)
        hbox2.Add(bookmark3)
        toolbar2.SetSizer(hbox2)
        vbox.Add(toolbar2, 0, wx.EXPAND)
        line = wx.StaticLine(panel)
        vbox.Add(line, 0, wx.EXPAND)

        panel.SetSizer(vbox)

        self.CreateStatusBar()
        self.Centre()
        self.Show(True)

app = wx.App(0)
Browser(None, -1, 'Browser')
app.MainLoop()
```

La pregunta es, como crear cuadro combinado redimensionable, que es usado tanto Firefox como en Opera? Nosotros no podemos usar un *wx.Toolbar*. Esto no es posible para crear tal funcionalidad con un *wx.Toolbar*. Confirmado con Robin Dunn. Así debemos hacer una solución.

```
toolbar1 = wx.Panel(panel, -1, size=(-1, 40))
```

El truco es simple. Creamos un *wx.Panel* plano.

```
hbox1 = wx.BoxSizer(wx.HORIZONTAL)
...
hbox1.Add(back)
hbox1.Add(forward)
hbox1.Add(refresh)
```

creamos un dimensionador horizontal y agregamos todos los botones necesarios.

```
hbox1.Add(address, 1, wx.TOP, 4)
```

Entonces agregamos el cuadro combinado al dimensionador. Este tipo de cuadro combinado es usualmente llamado una barra de direcciones. Aviso, que éste es el componente solamente, que tiene la proporción fijada a 1. Esto fue necesarios para hacerlo redimensionable.

La segunda barra de herramientas fue creada de manera similar. Las barras de Herramientas son separadas por una línea. Primero pensé, éste fue algún tipo de borde de panel. Prové todos los bordes posibles, pero no fue lo qué esperaba.

```
line = wx.StaticLine(panel)
```

Entonces lo tengo estudiado. Esto es una simple línea estática!

Algunas veces, debemos crear una solución, para lo cual no tenemos un componente adecuado. Pero usando el simple sentido común, podemos fácilmente encontrar una forma.

En esta parte del tutorial de wxPython hemos creado algunos esqueletos de aplicaciones.

Creando componentes personalizados

Alguna vez usted miró una aplicación y se preguntó, como un GUI particular item fue creado? Probablemente cada programador tiene quien lo imita. Entonces usted estaba buscando una lista de componentes provisto por su biblioteca GUI favorita. Pero usted no pudo encontrarlo. Los kits de herramientas usualmente proveen solamente los componentes más comunes tales como botones, componentes de texto, deslizadores etc. No todos pueden proveer todos los posible componentes.

Estos son en realidad dos tipos de kits de herramientas. Los kits de herramientas espartanos y los kits de herramientas muy pesados. El kits de herramientas FLTK es un tipo de kit de herramientas espartano. Este provee solamente componentes muy basicos y asume, que el programador se creará uno más complicado por sí mismo. wxPython es uno de los muy pesados. Éste tiene gran cantidad de componentes. Sin embargo no provee los componentes más especializados. Por ejemplo un componente medidor de velocidad, un componente que mida la capacidad de un CD a ser quemado (encontrados p.e. en nero). Los kits de herramientas no tienen usualmente componentes para gráficos.

Los programadores deben crear tales componentes por por si mismos. Aquellos lo hacen por usando las herramientas de dibujo provisto por el kit de herramientas. Estos son dos posibilidades. Un programador puede modificar o mejorar un componente existente. O él puede crear un componente

personalizado a partir de cero.

Aquí asumo, usted tiene leido el capítulo sobre el GDI.

Un componente hyperlink

El primer ejemplo creará un hyperlink. El componente hyperlink podrá ser basado sobre un componente *wx.lib.stattext.GenStaticText* existente.

```
#!/usr/bin/python
# link.py

import wx
from wx.lib.stattext import GenStaticText
import webbrowser


class Link(GenStaticText):
    def __init__(self, parent, id=-1, label='', pos=(-1, -1),
        size=(-1, -1), style=0, name='Link', URL=''):

        GenStaticText.__init__(self, parent, id, label, pos, size, style,
name)

        self.url = URL

        self.font1 = wx.Font(9, wx.SWISS, wx.NORMAL, wx.BOLD, True,
'Verdana')
        self.font2 = wx.Font(9, wx.SWISS, wx.NORMAL, wx.BOLD, False,
'Verdana')

        self.SetFont(self.font2)
        self.SetForegroundColour('#0000ff')

        self.Bind(wx.EVT_MOUSE_EVENTS, self.OnMouseEvent)
        self.Bind(wx.EVT_MOTION, self.OnMouseEvent)


    def OnMouseEvent(self, event):
        if event.Moving():
            self.SetCursor(wx.StockCursor(wx.CURSOR_HAND))
            self.SetFont(self.font1)

        elif event.LeftUp():
            webbrowser.open_new(self.url)

        else:
            self.SetCursor(wx.NullCursor)
            self.SetFont(self.font2)

        event.Skip()


class HyperLink(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(220, 150))
```

```
        panel = wx.Panel(self, -1)
        Link(panel, -1, 'ZetCode', pos=(10, 60),
URL='http://www.zetcode.com')
        motto = GenStaticText(panel, -1, 'Knowledge solamente matters',
pos=(10, 30))
        motto.SetFont(wx.Font(9, wx.SWISS, wx.NORMAL, wx.BOLD, False,
'Verdana'))


        self.Centre()
        self.Show(True)

app = wx.App()
HyperLink(None, -1, 'A Hyperlink')
app.MainLoop()
```

Este componente hyperlink es basado en sobre un componente existente. En este ejemplo no sacamos nada, acabamos de usar un componente existente, el cual modificamos un poquito.

```
from wx.lib.stattext import GenStaticText
import webbrowser
```

Aquí importamos el componente base desde la cual derivamos nuestra componente hyperlink y el módulo webbrowser. El módulo webbrowser es un módulo python estándar. Podríamos usarlo abrir enlaces en un navegador por omisión.

```
self.SetFont(self.font2)
self.SetForegroundColour('#0000ff')
```

La idea detras de creando un componente hyperlink es simple. Heredamos desde una clase base el componente *wx.lib.stattext.GenStaticText*. Así tenemos un componente de texto. Entonces lo modificamos un poquito para hacer un hyperlink fuera de este texto. Cambiamos el tipo de letra y el color del texto. Los hyperlinks son usualmente azules.

```
if event.Moving():
    self.SetCursor(wx.StockCursor(wx.CURSOR_HAND))
    self.SetFont(self.font1)
```

Si pasamos el puntero del ratón sobre el enlace, cambiamos el tipo de letra a subrayado y también cambia el puntero del ratón a un cursor con forma de mano.

```
elif event.LeftUp():
    webbrowser.open_new(self.url)
```

Si hacemos clic izquierdo sobre el enlace, abrimos el enlace en un navegador por omisión.

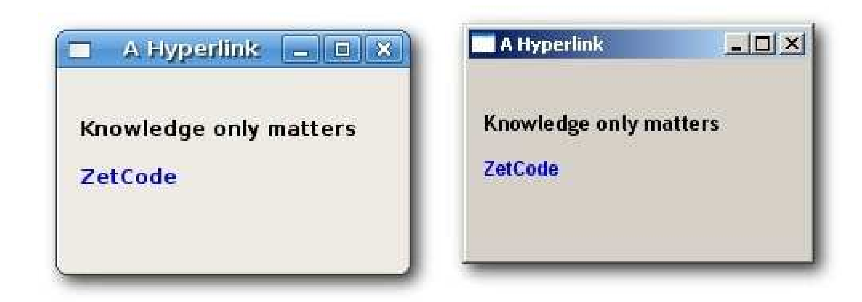


Figura: un componente Hyperlink

Componente quemador

Éste es un ejemplo de a componente, que creamos desde cero. Ponemos un wx.Panel sobre el fondo de la ventana y dibujamos el componente entero manualmente. Si usted tiene alguna vez que quemar un cd o un dvd, ya vió este tipo de componente.

Nota para usuarios de windows. Para evitar el parpadeo, usamos doble buffer.

```
#!/usr/bin/python
# burning.py

import wx

class Widget(wx.Panel):
    def __init__(self, parent, id):
        wx.Panel.__init__(self, parent, id, size=(-1, 30),
style=wx.SUNKEN_BORDER)
        self.parent = parent
        self.font = wx.Font(9, wx.FONTFAMILY_DEFAULT, wx.FONTSTYLE_NORMAL,
            wx.FONTWEIGHT_NORMAL, False, 'Courier 10 Pitch')


        self.Bind(wx.EVT_PAINT, self.OnPaint)
        self.Bind(wx.EVT_SIZE, self.OnSize)


    def OnPaint(self, event):
        num = range(75, 700, 75)
        dc = wx.PaintDC(self)
        dc.SetFont(self.font)
        w, h = self.GetSize()

        self.cw = self.parent.GetParent().cw

        step = int(round(w / 10.0))

        j = 0

        till = (w / 750.0) * self.cw
        full = (w / 750.0) * 700


        if self.cw >= 700:
            dc.SetPen(wx.Pen('#FFFFB8'))
            dc.SetBrush(wx.Brush('#FFFFB8'))
            dc.DrawRectangle(0, 0, full, 30)
            dc.SetPen(wx.Pen('#ffafaf'))
            dc.SetBrush(wx.Brush('#ffafaf'))
            dc.DrawRectangle(full, 0, till-full, 30)
        else:
            dc.SetPen(wx.Pen('#FFFFB8'))
            dc.SetBrush(wx.Brush('#FFFFB8'))
            dc.DrawRectangle(0, 0, till, 30)
```

```
        dc.SetPen(wx.Pen('#5C5142'))
        for i in range(step, 10*step, step):
            dc.DrawLine(i, 0, i, 6)
            width, height = dc.GetTextExtent(str(num[j]))
            dc.DrawText(str(num[j]), i-width/2, 8)
            j = j + 1

    def OnSize(self, event):
        self.Refresh()


class Burning(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(330, 200))

        self.cw = 75

        panel = wx.Panel(self, -1)
        CenterPanel = wx.Panel(panel, -1)
        self.sld = wx.Slider(CenterPanel, -1, 75, 0, 750, (-1, -1), (150,
-1), wx.SL_LABELS)

        vbox = wx.BoxSizer(wx.VERTICAL)
        hbox = wx.BoxSizer(wx.HORIZONTAL)
        hbox2 = wx.BoxSizer(wx.HORIZONTAL)
        hbox3 = wx.BoxSizer(wx.HORIZONTAL)

        self.wid = Widget(panel, -1)
        hbox.Add(self.wid, 1, wx.EXPAND)

        hbox2.Add(CenterPanel, 1, wx.EXPAND)
        hbox3.Add(self.sld, 0, wx.TOP, 35)

        CenterPanel.SetSizer(hbox3)

        vbox.Add(hbox2, 1, wx.EXPAND)
        vbox.Add(hbox, 0, wx.EXPAND)


        self.Bind(wx.EVT_SCROLL, self.OnScroll)

        panel.SetSizer(vbox)

        self.sld.SetFocus()

        self.Centre()
        self.Show(True)

    def OnScroll(self, event):
        self.cw = self.sld.GetValue()
        self.wid.Refresh()


app = wx.App()
Burning(None, -1, 'Burning widget')
```

```
app.MainLoop()
```

Todo el código importante reside en el método *OnPaint()* del componente de la clase. Este componente muestra graficamente la capacidad total de un medio y el espacio libre disponible para nosotros. El componente es controlado por un control deslizante. El valor mínimo de nuestra componente personalizado es 0, el máximo es 750. Si alcanzamos el valor 700, comenzamos a dibujar en color rojo. Esto normalmente indica sobre quemado.

```
w, h = self.GetSize()
self.cw = self.parent.GetParent().cw
...
till = (w / 750.0) * self.cw
full = (w / 750.0) * 700
```

Dibujamos el componente dinamicamente. Cuando mayor es la ventana, mayor es el componente quemador. Y vice versa. que es el por qué debemos calcular el tamaño del *wx.Panel* sobre los cuales dibujamos el componente personalizado. El parámetro hasta determina el tamaño total a ser dibujado. Este valor viene desde el control deslizante. Esto es una proporción del área entera. El parámetro full determina el punto, donde empezamos a dibujar en color rojo. Tenga en cuenta que usamos de aritmética de punto flotante. Éste es para lograr una mayor precision.

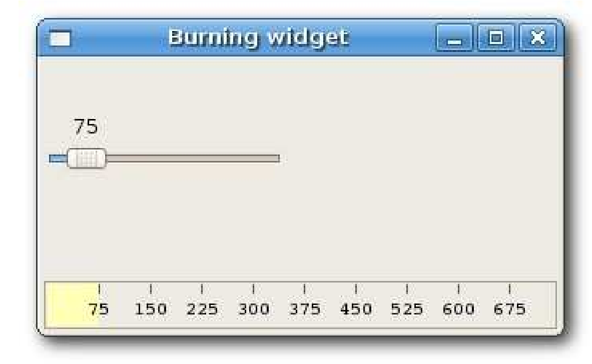
El dibujo actual consiste de los tres pasos. Dibujamos el rectángulo amarillo o rojo y amarillo. Entonces dibujamos las líneas verticales, las cuales dividen el componente en varias partes. Finalmente, dibujamos los números, los cuales indican la capacidad del medio.

```
def OnSize(self, event):
    self.Refresh()
```

Cada vez que la ventana es redimensionada, refrescamos el componente. Esto causa que el componente se vuelva a pintar por sí mismo.

```
def OnScroll(self, event):
    self.cw = self.sld.GetValue()
    self.wid.Refresh()
```

Si desplazamos el pulgar del control deslizante, tomamos el valor actual y lo guardamos dentro del parámetro *self.cw*. Este valor es usado, cuando el componente quemador es dibujado. Entonces causamos que el componente sea redibujado.



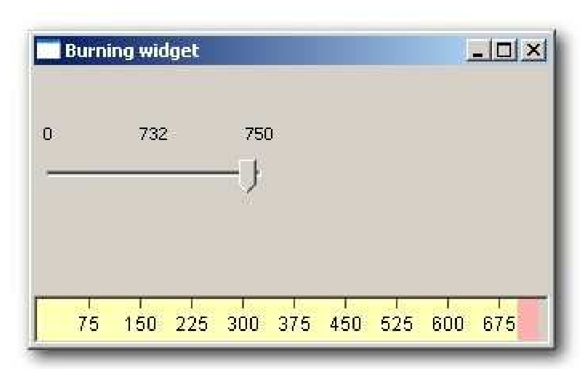


Figura: componenteQuemador

El componente CPU

Estas aplicaciones son sistema que miden recursos del sistema. La temperatura, memoria y consumo de CPU etc. Por mostrar un simple texto tal como CPU 54% usted probablemente no va a impresionar

sus usuarios. Componentes especializados son creados para hacer las aplicaciones más atractivas.

El siguientes componente es a menudo usado en aplicaciones de sistema.

Nota para usuarios de windows. Para evitar el parpadeo, usamos doble buffer. Cambia el tamaño de las aplicaciones y el ancho del control deslizante.

```
#!/usr/bin/python
# cpu.py

import wx


class CPU(wx.Panel):
    def __init__(self, parent, id):
        wx.Panel.__init__(self, parent, id, size=(80, 110))

        self.parent = parent

        self.SetBackgroundColour('#000000')


        self.Bind(wx.EVT_PAINT, self.OnPaint)


    def OnPaint(self, event):

        dc = wx.PaintDC(self)

        dc.SetDeviceOrigin(0, 100)
        dc.SetAxisOrientation(True, True)

        pos = self.parent.GetParent().GetParent().sel
        rect = pos / 5

        for i in range(1, 21):
            if i > rect:
                dc.SetBrush(wx.Brush('#075100'))
                dc.DrawRectangle(10, i*4, 30, 5)
                dc.DrawRectangle(41, i*4, 30, 5)
            else:
                dc.SetBrush(wx.Brush('#36ff27'))
                dc.DrawRectangle(10, i*4, 30, 5)
                dc.DrawRectangle(41, i*4, 30, 5)


class CPUWidget(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(190, 140))

        self.sel = 0

        panel = wx.Panel(self, -1)
        centerPanel = wx.Panel(panel, -1)

        self.cpu = CPU(centerPanel, -1)
```

```
        hbox = wx.BoxSizer(wx.HORIZONTAL)
        self.slider = wx.Slider(panel, -1, self.sel, 0, 100, (-1, -1), (25, 
90), 
                wx.VERTICAL | wx.SL_LABELS | wx.SL_INVERSE)
        self.slider.SetFocus()

        hbox.Add(centerPanel, 0,  wx.LEFT | wx.TOP, 20)
        hbox.Add(self.slider, 0, wx.LEFT | wx.TOP, 23)


        self.Bind(wx.EVT_SCROLL, self.OnScroll)

        panel.SetSizer(hbox)

        self.Centre()
        self.Show(True)


    def OnScroll(self, event):
        self.sel = event.GetInt()
        self.cpu.Refresh()


app = wx.App()
CPUWidget(None, -1, 'cpu')
app.MainLoop()
```

Crear este componente es bastante simple. Creamos un panel negro. Entonces dibujamos pequeños rectángulos sobre este panel. El color de los rectángulos depende del valor del control deslizante. El color pueden ser verde oscuro o verde brillante.

```
dc.SetDeviceOrigin(0, 100)
dc.SetAxisOrientation(True, True)
```

Aquí cambiamos el sistema de coordenadas por omisión a cartesianas. Esto es hacer el dibujo intuitivo.

```
pos = self.parent.GetParent().GetParent().sel
rect = pos / 5
```

Aquí tomamos el valor del dimensionador. Tenemos 20 rectángulos en cada columna. El control deslizante tiene 100 números. El parámetro rect hace una conversión desde los valores del control deslizante dentro de rectángulos, que podrá ser dibujado en color verde brillante.

```
for i in range(1, 21):
    if i > rect:
        dc.SetBrush(wx.Brush('#075100'))
        dc.DrawRectangle(10, i*4, 30, 5)
        dc.DrawRectangle(41, i*4, 30, 5)
    else:
        dc.SetBrush(wx.Brush('#36ff27'))
        dc.DrawRectangle(10, i*4, 30, 5)
        dc.DrawRectangle(41, i*4, 30, 5)
```

Aquí dibujamos 40 rectángulos, 20 en cada columna. Si el número del rectángulo que está siendo dibujado es mayor que el valor rect convertido, lo dibujamos en un color verde oscuro. De otra manera

en verde brillante.

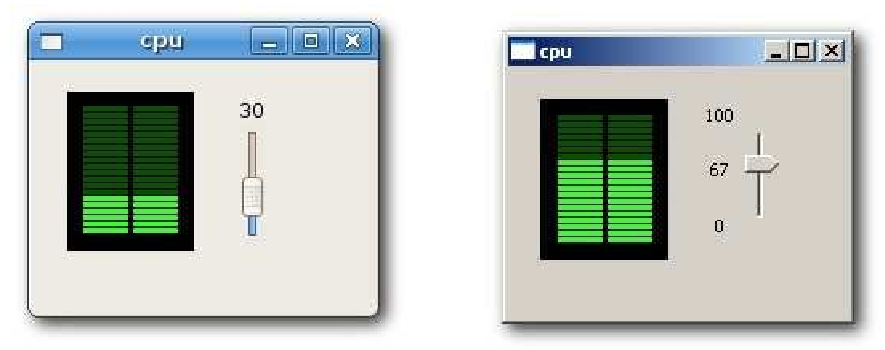


Figura: componentecpu

En este capítulo, hemos creados componente personalizados en wxPython.

El GDI

El **GDI** (**Interfaz de Dispositivo Gráfico**) es un interfaz para trabajar con gráficos. Esto es usado para interactuar con dispositivos gráficos tales como el monitor, la impresora o un archivo. El GDI permite a los programadores mostrar datos sobre una pantalla o la impresora sin tener que preocuparse acerca del detalles de a dispositivo particular. El GDI aisla al programador del hardware.

Dese el punto de vista del programador, el GDI es un grupo de clases y métodos para trabajar con gráficos. El GDI consiste de las 2D Vector Graphics, Fonts y Images.

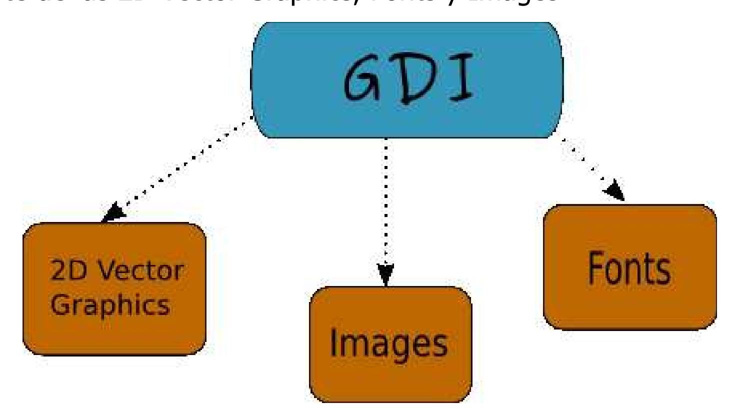


Figura: el GDI estructura

Para empezar a dibujar gráficos, debemos crear un objeto **contexto de dispositivo** (**DC**). En wxPython el contexto de dispositivo es llamado **wx.DC**. La documentación define un wx.DC como un contexto de dispositivo (canvas o lienzo) sobre el cual gráficos y texto pueden ser dibujados. Éste representa el número de dispositivo en una forma genérica. La misma pieza de código puede escribir en diferentes tipos de dispositivos. Ya sea una pantalla o una impresora. El wx.DC no pretende estar para ser usado directamente. En su lugar un programador debe elegir una de las clases derivadas. Cada clase derivada se pretende usar bajo condiciones específicas.

Clase Derivadas de wx.DC

- wxBufferedDC

- wxBufferedPaintDC
- wxPostScriptDC
- wxMemoryDC
- wxPrinterDC
- wxScreenDC
- wxClientDC
- wxPaintDC
- wxWindowDC

El *wx.ScreenDC* es usado para dibujar donde quiera en la pantalla. El *wx.WindowDC* es usado si buscamos pintar sobre el ventana completa (Windows solamente). Este incluye decoración de las ventanas. El *wx.ClientDC* es usado para dibujar sobre el área cliente de una ventana. El área cliente es el área de una ventana sinsus decoraciones (título y borde). El *wx.PaintDC* es usado para dibujar sobre el área cliente también. Pero esta es una diferencia entre el *wx.PaintDC* y el *wx.ClientDC*. El *wx.PaintDC* debería ser usado solamente desde a *wx.PaintEvent*. El *wx.ClientDC* si acaso no se puede utilizar desde a *wx.PaintEvent*. El *wx.MemoryDC* es usado para dibujar gráficos sobre el mapa de bits. El *wx.PostScriptDC* es usado para escribir a archivos PostScript sobre cualquier plataforma. El *wx.PrinterDC* es usado para acceder a la impresora (Windows solamente).

Dibujando una línea simple

Nuestro primer ejemplo podrá dibujar una línea simple sobre el área cliente de una ventana.

```
DrawLine(int x1, int y1, int x2, int y2)
```

Este método dibuja una línea desde el primer punto al segundo. Excluyendo el segundo punto.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

"""
ZetCode wxPython tutorial

This program drea a line over the
frame window after of a while

author: Jan Bodnar
website: zetcode.com
last edited: November 2010
"""

import wx

class Example(wx.Frame):
    def __init__(self, parent, title):
        super(Example, self).__init__(parent, title=title,
            size=(250, 150))

        wx.FutureCall(2000, self.DrawLine)

        self.Centre()
        self.Show()

    def DrawLine(self):
        dc = wx.ClientDC(self)
        dc.DrawLine(50, 60, 190, 60)
```

```
if __name__ == '__main__':
    app = wx.App()
    Example(None, 'Line')
    app.MainLoop()
```

Dibujamos una línea sobre el marco de la ventana Después de transcurridos dos segundos.

```
wx.FutureCall(2000, self.DrawLine)
```

Llamamos el método DrawLine() después de que la ventana fue creada. Lo hacemos porque, cuando la ventana es creada, ésta es dibujada. Todos nuestras dibujos podría haber sido por lo tanto perdido. Podemos comenzar el dibujo después de que la ventana fue creada. Éste es la razón, por qué llamamos el método wx.FutureCall().

```
def DrawLine(self):
    dc = wx.ClientDC(self)
    dc.DrawLine(50, 60, 190, 60)
```

Creamos un contexto de dispositivo wx.ClientDC. El único parámetro es la ventana sobre la cual buscamos dibujar. En nuestro caso ésta es self, la cual es una referencia a nuestro componente wx.Frame. Llamamos el método DrawLine() del contexto de dispositivo. Esta llamada en realidad dibuja una línea sobre nuestra ventana.

Éste es muy importante para comprender el siguiente comportamiento. Si redimensionamos la ventana, la línea podrá desaparecer. ¿Por qué es pasa esto? Toda la ventana es redibujada, si ésta es redimensionada. La línea también es redibujada, si ésta es maximizada. La ventana es también redibujada, si la tapamos por otra ventana y la destapamos después. La ventana es dibujado por su estado por omisión y nuestras línea se pierde. Tenemos que dibujar la línea cada vez que la ventana es redimensionada. El solution es el wx.PaintEvent. Este evento es disparado cada vez, la ventana es redibujada. Podríamos dibujar nuestras líneas en el interior de un método que podrá ser enganchado al evento pintar.

El siguientes ejemplo muestra como esto es hecho.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

"""
ZetCode wxPython tutorial

Este program dibuja una línea en
a evento pintar

author: Jan Bodnar
website: zetcode.com
last edited: November 2010
"""

import wx

class Example(wx.Frame):
    def __init__(self, parent, title):
        super(Example, self).__init__(parent, title=title,
            size=(250, 150))

        self.Bind(wx.EVT_PAINT, self.OnPaint)
```

```
        self.Centre()
        self.Show()

    def OnPaint(self, e):
        dc = wx.PaintDC(self)
        dc.DrawLine(50, 60, 190, 60)

if __name__ == '__main__':
    app = wx.App()
    Example(None, 'Line')
    app.MainLoop()
```

Dibujamos la misma línea. En este momento en reacción al evento pintar.

```
self.Bind(wx.EVT_PAINT, self.OnPaint)
```

Aquí vinculamos el método OnPaint al evento wx.PaintEvent. Esto significa que, que cada vez que nuestra ventana es repintada, llamamos al método OnPaint. Ahora la línea no podran desaparecer, si redimensionamos nuestras ventanas (taparla, maximizarla).

```
dc = wx.PaintDC(self)
```

Aviso, que en este momento estamos usado el contexto de dispositivo wx.PaintDC.

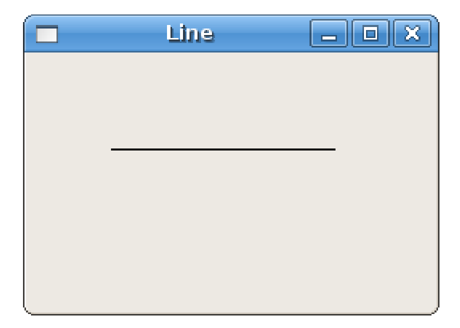


Figura: dibujo de una línea

Vector Gráficos Vectoriales y 2D

Estos son dos diferentes gráficos de computadora. Los gráficos **Vectoriales** y **Raster**. Los gráficos Raster representan imágenes como una colección de píxeles. Los gráficos Vectoriales los usamos en primitivas geométricas tales como puntos, líneas, curvas o polígonos para representar imágenes. Estas primitivas son creadas usando equaciones matemáticas.

Ambos tipos de gráficos de computadora tienen ventajas y desventajas. Las ventajas de los gráficos vectoriales sobre los raster son:

- menor tamaño
- habilidad para hacer zoom indefinidamente
- ya sea moviendo, escalando, rellenando o rotando no se degrada la calidad de una imagen

Tipos de primitivas

- puntos
- líneas
- polilíneas

- polígonos
- círculos
- elipses
- Splines

Atributos de los contextos de dispositivo

Atributo	Objeto	Valor por Omisión	Método para Tomar	Método para Fijar
Brush	wx.Brush	wx.WHITE_BRUSH	wx.Brush GetBrush()	SetBrush(wx.Brush brush)
Pen	wx.Pen	wx.BLACK_PEN	wx.Pen GetPen()	SetPen(wx.Pen pen)
modo de mapeo	-	wx.MM_TEXT	int GetMapMode()	SetMapMode(int mode)
BackgroundMode	-	wx.TRANSPARENT	int GetBackgroundMode()	SetBackgroundMode(int mode)
Text background colour	wx.Colour	wx.WHITE	wx.Colour GetTextBackground()	SetTextBackground(wx.Colour colour)
Text foreground colour	wx.Colour	wx.BLACK	wx.Colour GetTextForeground()	SetTextForeground(wx.Colour colour)

Elementos Basicos

En las siguientes líneas vamos a introducir varios objetos elementales. Colours (colores), Brushes (cepillos), Pens (plumas), Joins (juntas), Caps (Capas), Gradients (Gradientes).

Colours (Colores)

Un color es un objeto representando una combinación de valores de intensidad Rojo, Verde, y Azul (RGB). Valores RGB Válidos son en el rango de 0 a 255. Estos son tres caminos para establecer colores. Podemos crear un objeto wx.Colour, usamos un nombre de color predefinido o usamos una cadena de valor hex. *wx.Colour(0,0,255)*, *'BLUE'*, *'#0000FF'*. Estos tres notaciones produce el mismo color.

Una herramienta perfecta para trabajar con colores puede ser encontrada en el sitio Web colorjack.com. o podemos usar una herramienta como Gimp.

Tenemos también una lista de nombres de colores predefinidos que podemos usar en nuestros programas.

La base de datos de Colores Estándar

AQUAMARINE	BLACK	BLUE	BLUE VIOLET	BROWN
CADET BLUE	CORAL	CORNFLOWER BLUE	CYAN	DARK GREY
verde oscuro	DARK OLIVE GREEN	DARK ORCHID	DARK SLATE BLUE	DARK SLATE GREY
DARK TURQUOISE	DIM GREY	FIREBRICK	FOREST GREEN	GOLD
GOLDENROD	GREY	GREEN	GREEN YELLOW	INDIAN RED
KHAKI	LIGHT BLUE	LIGHT GREY	LIGHT STEEL BLUE	LIME GREEN
MAGENTA	MAROON	MEDIUM AQUAMARINE	MEDIUM BLUE	MEDIUM FOREST GREEN
MEDIUM GOLDENROD	MEDIUM ORCHID	MEDIUM SEA GREEN	MEDIUM SLATE BLUE	MEDIUM SPRING GREEN
MEDIUM TURQUOISE	MEDIUM VIOLET	MIDNIGHT BLUE	NAVY	ORANGE

	RED			
ORANGE RED	ORCHID	PALE GREEN	PINK	PLUM
PURPLE	RED	SALMON	SEA GREEN	SIENNA
SKY BLUE	SLATE BLUE	SPRING GREEN	STEEL BLUE	TAN
THISTLE	TURQUOISE	VIOLET	VIOLET RED	WHEAT
WHITE	YELLOW	YELLOW GREEN		

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

"""
ZetCode wxPython tutorial

Este programa dibuja nueve colores sobre la ventana

author: Jan Bodnar
website: zetcode.com
last edited: November 2010
"""

import wx

class Example(wx.Frame):
    def __init__(self, parent, title):
        super(Example, self).__init__(parent, title=title,
            size=(350, 280))

        self.Bind(wx.EVT_PAINT, self.OnPaint)

        self.Centre()
        self.Show()


    def OnPaint(self, e):
        dc = wx.PaintDC(self)
        dc.SetPen(wx.Pen('#d4d4d4'))

        dc.SetBrush(wx.Brush('#c56c00'))
        dc.DrawRectangle(10, 15, 90, 60)

        dc.SetBrush(wx.Brush('#1ac500'))
        dc.DrawRectangle(130, 15, 90, 60)

        dc.SetBrush(wx.Brush('#539e47'))
        dc.DrawRectangle(250, 15, 90, 60)

        dc.SetBrush(wx.Brush('#004fc5'))
        dc.DrawRectangle(10, 105, 90, 60)

        dc.SetBrush(wx.Brush('#c50024'))
        dc.DrawRectangle(130, 105, 90, 60)

        dc.SetBrush(wx.Brush('#9e4757'))
```

```
        dc.DrawRectangle(250, 105, 90, 60)

        dc.SetBrush(wx.Brush('#5f3b00'))
        dc.DrawRectangle(10, 195, 90, 60)

        dc.SetBrush(wx.Brush('#4c4c4c'))
        dc.DrawRectangle(130, 195, 90, 60)

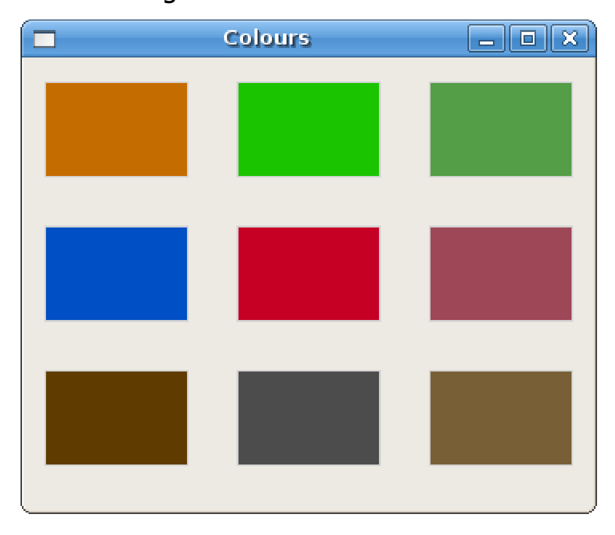
        dc.SetBrush(wx.Brush('#785f36'))
        dc.DrawRectangle(250, 195, 90, 60)


if __name__ == '__main__':
    app = wx.App()
    Example(None, 'Colours')
    app.MainLoop()
```

Dibujamos nueve rectángulos y los rellenamos con diferentes colores.

```
dc.SetBrush(wx.Brush('#c56c00'))
dc.DrawRectangle(10, 15, 90, 60)
```

Especificamos el color del pincel en notación hexadecimal. El pincel es el relleno de fondo de la forma (o figura). Entonces dibujamos el rectángulo.



wx.Pen

la pluma (o la pluma) es un objeto gráficos elementales. Esto es usado para dibujar líneas, curvas y las líneas del borde de de rectángulos, elipses, polígonos o otras figuras.

```
wx.Pen(wx.Colour colour, width=1, style=wx.SOLID)
```

El constructor de *wx.Pen* tiene tres parámetros. Colour, width y style. Sigue una lista de posible estilos de pluma (pen).

Estilos de plumas

- wx.SOLID
- wx.DOT
- wx.LONG_DASH
- wx.SHORT_DASH

- wx.DOT_DASH
- wx.TRANSPARENT

```
#!/usr/bin/python
# pens.py

import wx

class Pens(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(350, 190))

        self.Bind(wx.EVT_PAINT, self.OnPaint)

        self.Centre()
        self.Show(True)

    def OnPaint(self, event):
        dc = wx.PaintDC(self)

        dc.SetPen(wx.Pen('#4c4c4c', 1, wx.SOLID))
        dc.DrawRectangle(10, 15, 90, 60)

        dc.SetPen(wx.Pen('#4c4c4c', 1, wx.DOT))
        dc.DrawRectangle(130, 15, 90, 60)

        dc.SetPen(wx.Pen('#4c4c4c', 1, wx.LONG_DASH))
        dc.DrawRectangle(250, 15, 90, 60)

        dc.SetPen(wx.Pen('#4c4c4c', 1, wx.SHORT_DASH))
        dc.DrawRectangle(10, 105, 90, 60)

        dc.SetPen(wx.Pen('#4c4c4c', 1, wx.DOT_DASH))
        dc.DrawRectangle(130, 105, 90, 60)

        dc.SetPen(wx.Pen('#4c4c4c', 1, wx.TRANSPARENT))
        dc.DrawRectangle(250, 105, 90, 60)

app = wx.App()
Pens(None, -1, 'Pens')
app.MainLoop()
```

Si no especificamos a un pincel personalizado, es usado uno por omisión. El pincel por omisión es *wx.WHITE_BRUSH*. El perímetro de los rectángulos es dibujado por la pluma. El último no tiene borde. Éste es transparente, p.e. no visible.

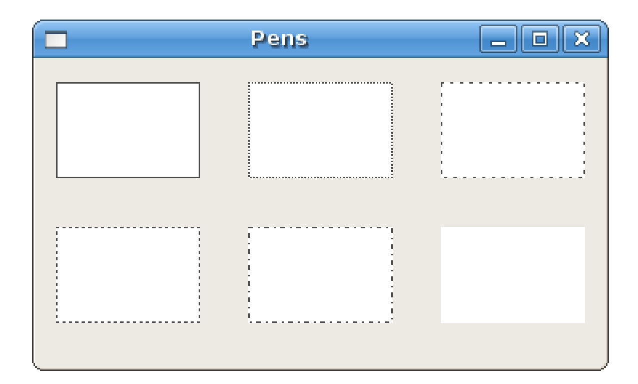


Figura: Pens (plumas)

Las uniones y las puntas

Un objeto pen tiene adicionalmente dos parámetros. *Join* y *Cap*. *Join* define como la unión entre líneas podrá ser dibujada. El estilo de *Join* tiene las siguientes opciones:

- wx.JOIN_MITER
- wx.JOIN_BEVEL
- wx.JOIN_ROUND

Cuando usando *wx.JOIN_MITER* el bordes exteriores de las líneas son extendidas. Aquellos se encuentran en un ángulo, y su área es rellenada. En muesca triangular *wx.JOIN_BEVEl* entre dos líneas es rellenada. En *wx.JOIN_ROUND* el arco circular entre los dos líneas es rellenada. El valor por omisión es *wx.JOIN_ROUND*.

El *Cap* define como la terminación de la línea podrá ser dibujadoapor la pluma. Las opciones son:

- wx.CAP_ROUND
- wx.CAP_PROJECTING
- wx.CAP_BUTT

El *wx.CAP_ROUND* podrá dibujar extremos redondeados. El *wx.CAP_PROJECTING* y el *wx.CAP_BUTT* podrá tanto dibujamos extremos cuadrados. El diferencia entre ellos es que el *wx.CAP_PROJECTING* podrá extender más allá del punto final por la mitad de los tamaños de línea. El *wx.CAP_ROUND* podrá extender más allá del punto final también.

```
#!/usr/bin/python
# joinscaps.py

import wx

class JoinsCaps(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(330, 300))

        self.Bind(wx.EVT_PAINT, self.OnPaint)

        self.Centre()
        self.Show(True)

    def OnPaint(self, event):
        dc = wx.PaintDC(self)

        pen = wx.Pen('#4c4c4c', 10, wx.SOLID)
```

```
        pen.SetJoin(wx.JOIN_MITER)
        dc.SetPen(pen)
        dc.DrawRectangle(15, 15, 80, 50)

        pen.SetJoin(wx.JOIN_BEVEL)
        dc.SetPen(pen)
        dc.DrawRectangle(125, 15, 80, 50)

        pen.SetJoin(wx.JOIN_ROUND)
        dc.SetPen(pen)
        dc.DrawRectangle(235, 15, 80, 50)

        pen.SetCap(wx.CAP_BUTT)
        dc.SetPen(pen)
        dc.DrawLine(30, 150,  150, 150)

        pen.SetCap(wx.CAP_PROJECTING)
        dc.SetPen(pen)
        dc.DrawLine(30, 190,  150, 190)

        pen.SetCap(wx.CAP_ROUND)
        dc.SetPen(pen)
        dc.DrawLine(30, 230,  150, 230)

        pen2 = wx.Pen('#4c4c4c', 1, wx.SOLID)
        dc.SetPen(pen2)
        dc.DrawLine(30, 130, 30, 250)
        dc.DrawLine(150, 130, 150, 250)
        dc.DrawLine(155, 130, 155, 250)

app = wx.App()
JoinsCaps(None, -1, 'Joins y Caps')
app.MainLoop()
```

```
 pen = wx.Pen('#4c4c4c', 10, wx.SOLID)
```

En orden para ver el varios estilos de *Join* y *Cap*, necesitamos fijar el ancho de la pluma para que sea mayor a 1.

```
 dc.DrawLine(150, 130, 150, 250)
 dc.DrawLine(155, 130, 155, 250)
```

Atención a los dos líneas verticales adjuntas. La distancia entre ellos es 5px. Éste es exactamente la mitad del ancho de la pluma corriente.

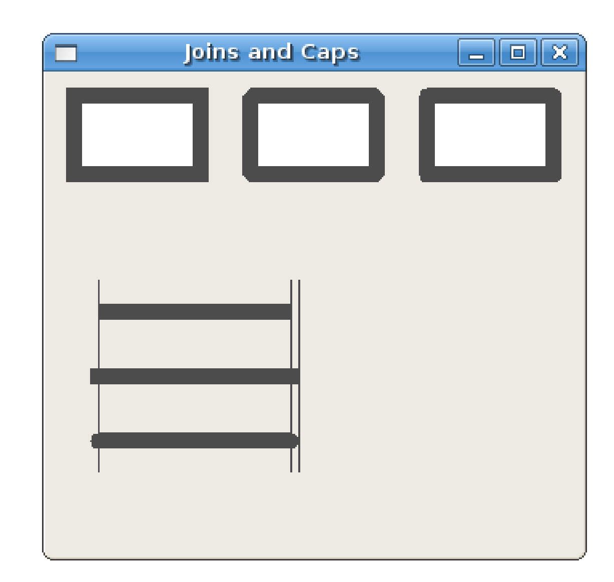


Figura: Joins y Caps

Gradientes

En gráficos de computadora, gradiente es una mezcla suave de sombras desde lo claro a lo oscuro o desde un color a otro. En programas de dibujo 2D y programas de dibujo, los gradientes son usados para crear fondos de colores y efectos especiales también como para simular luces y sombras. (answers.com)

```
  GradientFillLinear(wx.Rect rect, wx.Colour initialColour, wx.Colour
destColour, int nDirection=wx.EAST)
```

Este método rellena el área especificada por un *rect* con un gradiente lineal, empezando desde *initialColour* y eventualmente desvaneciendose hasta *destColour*. El parámetro nDirection especifica la dirección en la que el color cambia, el valor por omisión es wx.EAST.

```
#!/usr/bin/python
# gradients.py

import wx

class Gradients(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(220, 260))

        self.Bind(wx.EVT_PAINT, self.OnPaint)

        self.Centre()
        self.Show(True)

    def OnPaint(self, event):
        dc = wx.PaintDC(self)

        dc.GradientFillLinear((20, 20, 180, 40), '#ffec00', '#000000',
wx.NORTH)
        dc.GradientFillLinear((20, 80, 180, 40), '#ffec00', '#000000',
wx.SOUTH)
        dc.GradientFillLinear((20, 140, 180, 40), '#ffec00', '#000000',
wx.EAST)
        dc.GradientFillLinear((20, 200, 180, 40), '#ffec00', '#000000',
```

```
wx.WEST)


app = wx.App()
Gradients(None, -1, 'Gradients')
app.MainLoop()
```



Figura: Gradients

wx.Brush

Brush es un objeto de gráficos elementales. Esto es usado para pintar el fondo de gráficos figuras, tales como rectángulos, elipses o polígonos.

```
wx.Brush(wx.Colour colour, style=wx.SOLID)
```

El constructor del *wx.Brush* acepta dos parámetros. Colour y style. La siguiente es una lista de posible estilos de pinceles.

Estilos de pinceles

- wx.SOLID
- wx.STIPPLE
- wx.BDIAGONAL_HATCH
- wx.CROSSDIAG_HATCH
- wx.FDIAGONAL_HATCH
- wx.CROSS_HATCH
- wx.HORIZONTAL_HATCH
- wx.VERTICAL_HATCH
- wx.TRANSPARENT

```
#!/usr/bin/python
# brushes.py

import wx

class Brush(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(350, 280))

        self.Bind(wx.EVT_PAINT, self.OnPaint)
```

```
        self.Centre()
        self.Show(True)

    def OnPaint(self, event):
        dc = wx.PaintDC(self)

        dc.SetBrush(wx.Brush('#4c4c4c', wx.CROSS_HATCH))
        dc.DrawRectangle(10, 15, 90, 60)

        dc.SetBrush(wx.Brush('#4c4c4c', wx.SOLID))
        dc.DrawRectangle(130, 15, 90, 60)

        dc.SetBrush(wx.Brush('#4c4c4c', wx.BDIAGONAL_HATCH))
        dc.DrawRectangle(250, 15, 90, 60)

        dc.SetBrush(wx.Brush('#4c4c4c', wx.CROSSDIAG_HATCH))
        dc.DrawRectangle(10, 105, 90, 60)

        dc.SetBrush(wx.Brush('#4c4c4c', wx.FDIAGONAL_HATCH))
        dc.DrawRectangle(130, 105, 90, 60)

        dc.SetBrush(wx.Brush('#4c4c4c', wx.HORIZONTAL_HATCH))
        dc.DrawRectangle(250, 105, 90, 60)

        dc.SetBrush(wx.Brush('#4c4c4c', wx.VERTICAL_HATCH))
        dc.DrawRectangle(10, 195, 90, 60)

        dc.SetBrush(wx.Brush('#4c4c4c', wx.TRANSPARENT))
        dc.DrawRectangle(130, 195, 90, 60)


app = wx.App()
Brush(None, -1, 'Brushes')
app.MainLoop()
```

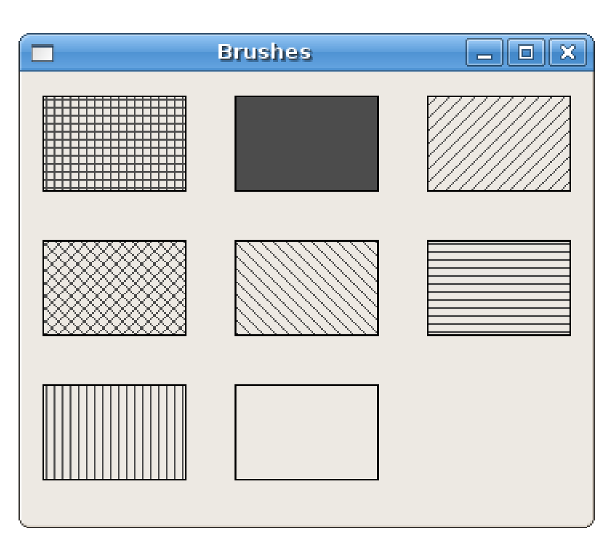


Figura: Pinceles

Patrones adaptados

No estamos restringidos a usar patrones predefinidos. Podemos fácilmente crear nuestros propios patrones personalizados.

```
wx.Brush BrushFromBitmap(wx.Bitmap stippleBitmap)
```

Este método crea un pincel personalizado desde el mapa de bits.

```
#!/usr/bin/python
# custompatterns.py

import wx

class CustomPatterns(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(350, 280))

        self.Bind(wx.EVT_PAINT, self.OnPaint)

        self.Centre()
        self.Show(True)

    def OnPaint(self, event):
        dc = wx.PaintDC(self)

        dc.SetPen(wx.Pen('#C7C3C3'))

        brush1 = wx.BrushFromBitmap(wx.Bitmap('pattern1.png'))
        dc.SetBrush(brush1)
        dc.DrawRectangle(10, 15, 90, 60)

        brush2 = wx.BrushFromBitmap(wx.Bitmap('pattern2.png'))
        dc.SetBrush(brush2)
        dc.DrawRectangle(130, 15, 90, 60)

        brush3 = wx.BrushFromBitmap(wx.Bitmap('pattern3.png'))
        dc.SetBrush(brush3)
        dc.DrawRectangle(250, 15, 90, 60)

        brush4 = wx.BrushFromBitmap(wx.Bitmap('pattern4.png'))
        dc.SetBrush(brush4)
        dc.DrawRectangle(10, 105, 90, 60)

        brush5 = wx.BrushFromBitmap(wx.Bitmap('pattern5.png'))
        dc.SetBrush(brush5)
        dc.DrawRectangle(130, 105, 90, 60)

        brush6 = wx.BrushFromBitmap(wx.Bitmap('pattern6.png'))
        dc.SetBrush(brush6)
        dc.DrawRectangle(250, 105, 90, 60)

        brush7 = wx.BrushFromBitmap(wx.Bitmap('pattern7.png'))
        dc.SetBrush(brush7)
        dc.DrawRectangle(10, 195, 90, 60)

        brush8 = wx.BrushFromBitmap(wx.Bitmap('pattern8.png'))
```

```
        dc.SetBrush(brush8)
        dc.DrawRectangle(130, 195, 90, 60)

        brushr9 = wx.BrushFromBitmap(wx.Bitmap('pattern9.png'))
        dc.SetBrush(brushr9)
        dc.DrawRectangle(250, 195, 90, 60)


app = wx.App()
CustomPatterns(None, -1, 'Patrones adaptados')
app.MainLoop()
```

Tengo creado algunos pequeños bitmaps. Para esto he usado el Gimp. Estos bitmaps son rectángulos, usualmente alrededor de 40-150px.



Figura: Patrones adaptados

Primitivas básicas

Point

El más simple objeto geométrico es un punto. Esto es un punto claro sobre la ventana.

```
 DrawPoint(int x, int y)
```

Este método dibuja un punto en las coordenadas x, y.

```
#!/usr/bin/python
# points.py

import wx
import random

class Points(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(250, 150))

        self.Bind(wx.EVT_PAINT, self.OnPaint)
```

```
        self.Centre()
        self.Show(True)

    def OnPaint(self, event):
        dc = wx.PaintDC(self)

        dc.SetPen(wx.Pen('RED'))

        for i in range(1000):
            w, h = self.GetSize()
            x = random.randint(1, w-1)
            y = random.randint(1, h-1)
            dc.DrawPoint(x, y)


app = wx.App()
Points(None, -1, 'Points')
app.MainLoop()
```

Un punto simple sería difícil de ver. Así creamos 1000 puntos.

```
dc.SetPen(wx.Pen('RED'))
```

Aquí fijamos el color de la pluma a rojo.

```
w, h = self.GetSize()
x = random.randint(1, w-1)
```

Los puntos son distribuidos al azar alrededor del área cliente de la ventana. Aquellos son también distribuidos dinamicamente. Si redimensionamos la ventana, Los puntos podrán ser dibujados al azar sobre un nuevo tamaño de cliente. El método *randint(a, b)* retorna un entero aleatorio en el rango [a, b], p.e. incluyendo puntos.

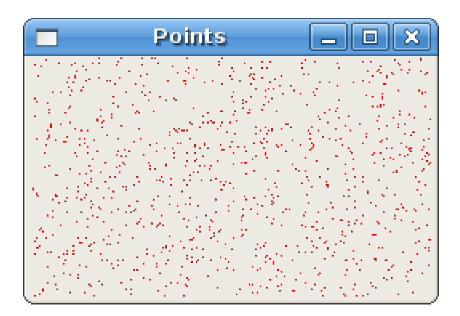


Figura: dibujo puntos

Figuras

Figuras son los objetos geométricos más sofisticados. Podríamos dibujar varios figuras geométricas en el siguiente ejemplo.

```
#!/usr/bin/python
# shapes.py

import wx
```

```
class Shapes(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(350, 300))

        self.Bind(wx.EVT_PAINT, self.OnPaint)

        self.Centre()
        self.Show(True)

    def OnPaint(self, event):
        dc = wx.PaintDC(self)

        dc.DrawEllipse(20, 20, 90, 60)
        dc.DrawRoundedRectangle(130, 20, 90, 60, 10)
        dc.DrawArc(240, 40, 340, 40, 290, 20)

        dc.DrawPolygon(((130, 140), (180, 170), (180, 140), (220, 110),
(140, 100)))
        dc.DrawRectangle(20, 120, 80, 50)
        dc.DrawSpline(((240, 170), (280, 170), (285, 110), (325, 110)))

        dc.DrawLines(((20, 260), (100, 260), (20, 210), (100, 210)))
        dc.DrawCircle(170, 230, 35)
        dc.DrawRectangle(250, 200, 60, 60)

app = wx.App()
Shapes(None, -1, 'Shapes')
app.MainLoop()
```

En nuestro ejemplo tenemos dibujado an elipse, un rectángulo redondeado, un arco, un rectángulo, un polígono, splines, líneas, un círculo y un cuadrado (de derecha a izquierda, de arriba abajo). un círculo es una especial tipo de Shapes y un cuadrado es un tipo especial de rectángulo.

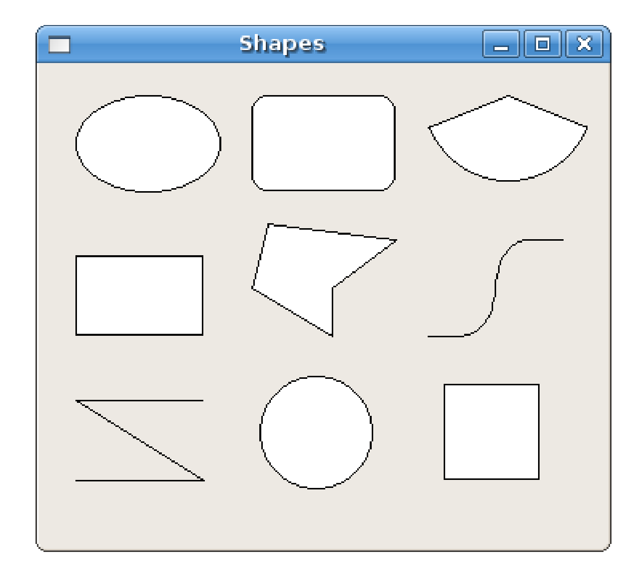


Figura: Figuras

Regiones

El contexto de dispositivo pueden ser dividido dentro de varias partes llamadas **Regiones**. Una región pueden ser una figura cualquiera. Una región pueden ser un simple rectángulo o circle. Con las operaciones *Union*, *Intersect*, *Substract* y *Xor* podemos crear regiones complejas a partir de simple. Las regiones son usadas para delineado, rellenando o recorte.

Podemos crear regiones de tres modos. La manera más fácil es crear una región rectangular. Regiones más complejas pueden ser creadas desde una lista de puntos o desde un mapa de bits.

```
wx.Region(int x=0, int y=0, int width=0, int height=0)
```

Este constructor crea una región rectangular.

```
wx.RegionFromPoints(list points, int fillStyle=wx.WINDING_RULE)
```

Este constructor crea una región poligonal. El parámetro *fillStyle* puede ser wx.WINDING_RULE o wx.ODDEVEN_RULE.

```
wx.RegionFromBitmap(wx.Bitmap bmp)
```

Las regiones más complejas pueden ser creadas con el método previo.

Antes de ir a las regiones, primero vamos a crear un ejemplo pequeño. Dividimos el tópico dentro de varias partes de modo que éste sea fácil para comprender. Usted puede encontrar lo como una buena idea para revisar vuestra matemática de la escuela. Aquí podemos encontrar un buen artículo.

```
#!/usr/bin/python
# lines.py

import wx
from math import hypot, sin, cos, pi

class Lines(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(450, 400))

        self.Bind(wx.EVT_PAINT, self.OnPaint)

        self.Centre()
        self.Show(True)

    def OnPaint(self, event):
        dc = wx.PaintDC(self)
        size_x, size_y = self.GetClientSizeTuple()
        dc.SetDeviceOrigin(size_x/2, size_y/2)

        radius = hypot(size_x/2, size_y/2)
        angle = 0

        while (angle < 2*pi):
            x = radius*cos(angle)
            y = radius*sin(angle)
            dc.DrawLinePoint((0, 0), (x, y))
            angle = angle + 2*pi/360

app = wx.App()
```

```
Lines(None, -1, 'Lines')
app.MainLoop()
```

En este ejemplo dibujamos 260 líneas desde el medio del área cliente. La distancia entre dos líneas es 1 grado. Creamos una figura interesante.

```
import wx
from math import hypot, sin, cos, pi
```

Necesitamos tres funciones matemáticas y una constante desde el módulo math.

```
dc.SetDeviceOrigin(size_x/2, size_y/2)
```

El método *SetDeviceOrigin()* crea un nuevo comienzo del sistema de coordenadas. Lo colocamos al medio dentro del área cliente. Por reposicionamiento del sistema de coordenadas, hacemos nuestro dibujo menos complicado.

```
radius = hypot(size_x/2, size_y/2)
```

Aquí tomamos la hipotenusa. Ésta es la línea más larga, podemos dibujar desde el medio del área cliente. Éste es el largo de la línea, que debería ser dibujado desde el comienzo dentro de la esquina de la ventana. De esta manera la mayoría del líneas no son dibujado completamente. Las partes superpuestas no son visible. Ver hipotenusa.

```
x = radius*cos(angle)
y = radius*sin(angle)
```

Estas son funciones paramétricas. Estas son usadas para encontrar los puntos [x, y] sobre la curva. Todos las 360 líneas son dibujadas desde el comienzo del sistema de coordenadas hasta los puntos sobre el circulo.

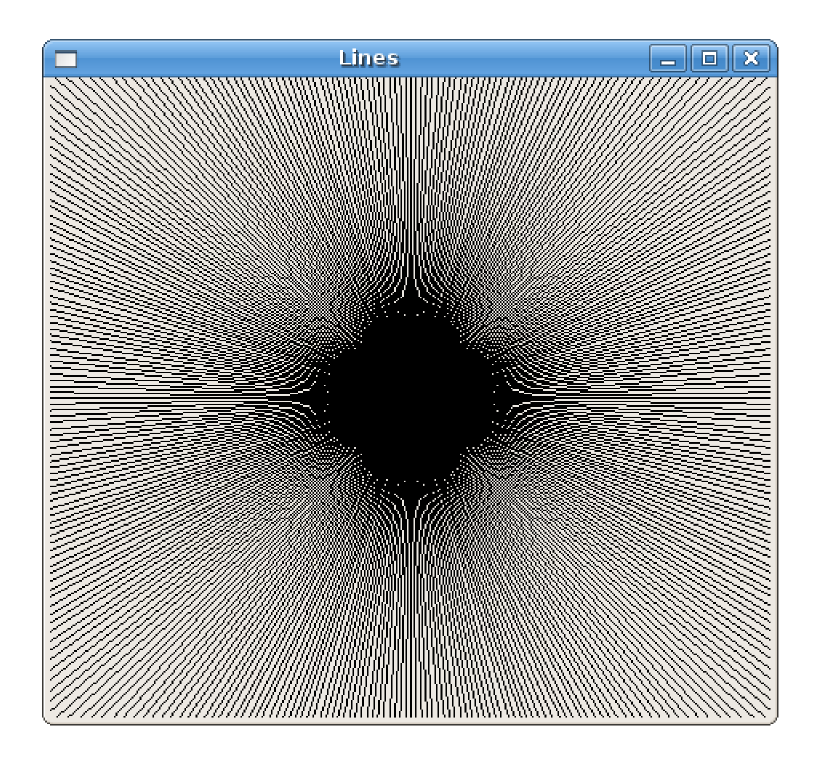


Figura: Líneas

Recorte (clipping)

El *Recorte* es restringir el dibujo a un cierto área. El recorte es usado en dos casos. Para crear efectos y to mejorar rendimiento de las aplicaciones. Restringimos el dibujo a cierta región con el método *SetClippingRegionAsRegion()*.

En el siguiente ejemplo podremos modificar y mejorar nuestras previo guión.

```
#!/usr/bin/python
# star.py

import wx
from math import hypot, sin, cos, pi

class Star(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(350, 300))

        self.Bind(wx.EVT_PAINT, self.OnPaint)

        self.Centre()
        self.Show(True)

    def OnPaint(self, event):
        dc = wx.PaintDC(self)

        dc.SetPen(wx.Pen('#424242'))
        size_x, size_y = self.GetClientSizeTuple()
        dc.SetDeviceOrigin(size_x/2, size_y/2)

        points = (((0, 85), (75, 75), (100, 10), (125, 75), (200, 85),
            (150, 125), (160, 190), (100, 150), (40, 190), (50, 125)))

        region = wx.RegionFromPoints(points)
        dc.SetClippingRegionAsRegion(region)

        radius = hypot(size_x/2, size_y/2)
        angle = 0

        while (angle < 2*pi):
            x = radius*cos(angle)
            y = radius*sin(angle)
            dc.DrawLinePoint((0, 0), (x, y))
            angle = angle + 2*pi/360

        dc.DestroyClippingRegion()


app = wx.App()
Star(None, -1, 'Star')
app.MainLoop()
```

Dibujamos de nuevo todos los el 360 líneas. Pero En este momento, solamente a porción del área del cliente es dibujada. La región que restringimos nuestros dibujo es a un objeto estrella.

```
 region = wx.RegionFromPoints(points)
 dc.SetClippingRegionAsRegion(region)
```

Creamos una región desde la lista de puntos con el método *wx.RegionFromPoins()*. El método *SetClippingRegionAsRegion()* restringe el dibujo a la región especificada. En nuestro caso, es un objeto estrella.

```
dc.DestroyClippingRegion()
```

Debemos destruir la región de recorte.

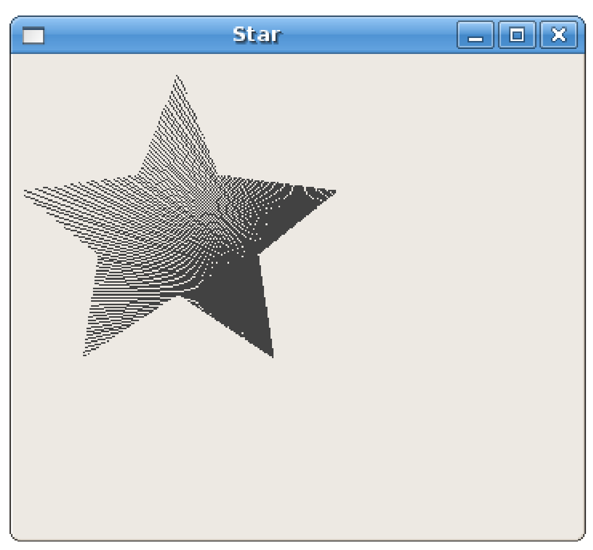


Figura: Star

Operaciones sobre Regiones

Las regiones pueden ser combinadas para crear figuras más complejas. podemos usar un conjunto de operaciones. *Union*, *Intersect*, *Substract* y *Xor*.

El siguientes ejemplo muestra todas los cuatro operaciones en acción.

```
#!/usr/bin/python
# operations.py

import wx

class Operations(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(270, 220))

        self.Bind(wx.EVT_PAINT, self.OnPaint)

        self.Centre()
        self.Show(True)

    def OnPaint(self, event):
        dc = wx.PaintDC(self)
        dc.SetPen(wx.Pen('#d4d4d4'))

        dc.DrawRectangle(20, 20, 50, 50)
        dc.DrawRectangle(30, 40, 50, 50)

        dc.SetBrush(wx.Brush('#ffffff'))
        dc.DrawRectangle(100, 20, 50, 50)
        dc.DrawRectangle(110, 40, 50, 50)
```

```
        region1 = wx.Region(100, 20, 50, 50)
        region2 = wx.Region(110, 40, 50, 50)
        region1.IntersectRegion(region2)
        rect = region1.GetBox()
        dc.SetClippingRegionAsRegion(region1)
        dc.SetBrush(wx.Brush('#ff0000'))
        dc.DrawRectangleRect(rect)
        dc.DestroyClippingRegion()

        dc.SetBrush(wx.Brush('#ffffff'))
        dc.DrawRectangle(180, 20, 50, 50)
        dc.DrawRectangle(190, 40, 50, 50)
        region1 = wx.Region(180, 20, 50, 50)
        region2 = wx.Region(190, 40, 50, 50)
        region1.UnionRegion(region2)
        dc.SetClippingRegionAsRegion(region1)
        rect = region1.GetBox()
        dc.SetBrush(wx.Brush('#fa8e00'))
        dc.DrawRectangleRect(rect)
        dc.DestroyClippingRegion()

        dc.SetBrush(wx.Brush('#ffffff'))
        dc.DrawRectangle(20, 120, 50, 50)
        dc.DrawRectangle(30, 140, 50, 50)
        region1 = wx.Region(20, 120, 50, 50)
        region2 = wx.Region(30, 140, 50, 50)
        region1.XorRegion(region2)
        rect = region1.GetBox()
        dc.SetClippingRegionAsRegion(region1)
        dc.SetBrush(wx.Brush('#619e1b'))
        dc.DrawRectangleRect(rect)
        dc.DestroyClippingRegion()

        dc.SetBrush(wx.Brush('#ffffff'))
        dc.DrawRectangle(100, 120, 50, 50)
        dc.DrawRectangle(110, 140, 50, 50)
        region1 = wx.Region(100, 120, 50, 50)
        region2 = wx.Region(110, 140, 50, 50)
        region1.SubtractRegion(region2)
        rect = region1.GetBox()
        dc.SetClippingRegionAsRegion(region1)
        dc.SetBrush(wx.Brush('#715b33'))
        dc.DrawRectangleRect(rect)
        dc.DestroyClippingRegion()

        dc.SetBrush(wx.Brush('#ffffff'))
        dc.DrawRectangle(180, 120, 50, 50)
        dc.DrawRectangle(190, 140, 50, 50)
        region1 = wx.Region(180, 120, 50, 50)
        region2 = wx.Region(190, 140, 50, 50)
        region2.SubtractRegion(region1)
        rect = region2.GetBox()
        dc.SetClippingRegionAsRegion(region2)
        dc.SetBrush(wx.Brush('#0d0060'))
        dc.DrawRectangleRect(rect)
        dc.DestroyClippingRegion()
```

```
app = wx.App()
Operations(None, -1, 'Operations')
app.MainLoop()
```

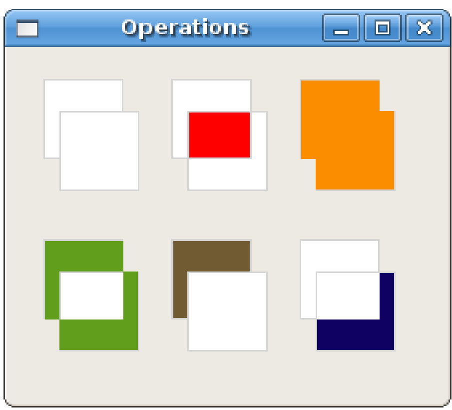


Figura: Conjunto de operaciones sobre Regiones

Modos de mapeo

Hablamos en Inglés, medimos en Métrico

El Inglés se convirtio en el lenguaje global para comunicación. Así como el sistema métrico se convirtió el sistema global para mediciones. De acuerdo al este artículo de wikipedia, hay solamente tres excepciones. USA, Liberia y Myanmar. Por ejemplo, los americanos usan Fahrenheits para medir la temperatura, gallones para el tanque de sus autos (carros) o libras para pesar las cargas.

Aunque nosotros en Europa usamos el sistema métrico, aquí todavía hay exepciones. USA es dominante en IT y estamos importando su estándares. Así we también decimos que tenemos un monitor de 17 Pulgadas. Los gráficos pueden ser puestos dentro de un archivo, mostrados en la pantalla de un monitor u otros dispositivos (cameras, videocameras, teléfonos moviles) o impresos con una la impresora. El tamaño del papel puede ser fijado en milímetros, puntos o puntos, la resolución de una pantalla es en pixels, la calidad de un texto es determinado por el número de puntos por pulgada. Tenemos también puntos, bits o muestras. Éste es una de las razones por la cual tenemos unidades **lógicas** y de **dispositivo**.

Unidades Logicas y de dispositivo

Si dibujamos texto o primitivas geométricas sobre el área cliente, las posicionamos usando unidades lógicas.

DrawText(string text, int x, int y)

Si buscamos dibujar algunos textos, proveemos el parámetro de texto y las posiciones x, y. x, y están en unidades lógicas. El dispositivo entonces dibuja el texto en unidades de dispositivo. Las unidades lógicas y de dispositivo pueden ser las mismas, o aquellas puede diferir. Las unidades lógicas son usadas por personas (milímetros), unidades de dispositivo are son nativas de un *dispositivo particular*. Por ejemplo una unidad nativa de dispositivo para una pantalla es el pixel. La unidad nativa para el dispositivo *HEWLETT PACKARD LaserJet 1022* es 1200 dpi. (puntos por pulgada).

Hasta el momento tenemos hablared acerca de varios midenment unidades. El **modo de mapeo** del dispositivo es una forma para convertir unidades lógicas a unidades de dispositivo. wxPython tiene los siguientes modos de mapeo:

Modo de Mapeo	Unidades lógicas
wx.MM_TEXT	1 pixel
wx.MM_METRIC	1 milímetro
wx.MM_LOMETRIC	1/10 de milímetro
wx.MM_POINTS	1 point, 1/72 de an inch
wx.MM_TWIPS	1/20 de un punto o 1/1440 de una pulgada

El modo de mapeo por omisión es wx.MM_TEXT. En este modo, las unidades lógicas son las mismas que las unidades de dispositivo. Cuando las personas posicionan objeto sobre una pantalla o diseñan una página Web, aquellos piensan usualmente en pixels. Los diseñadores Web crean páginas de tres columas y estas columnas son fijadas en pixels. El menor común denominador para una página es a menudo 800 px etc. Esto pensando como es natural que sabemos que nuestros monitores tienen p.e. 1024x768 pxs. No estamos para hacer conversiones, más bien estamos acostumbrados a pensar en pixels. Si buscamos dibujar a estructura en milímetros, podemos usar los dos modos métricos de mapeo. Dibujando directamente en milímetros es muy gruesa para una pantalla, es por eso que tenemos el modo de mapeo wx.MM_LOMETRIC.

SetMapMode(int mode)

Para fijar un modo de mapeo diferente, usamos el método *SetMapMode()*.

Ejemplo ruler1

El primer ejemplo ruler1 podrá medir pantalla objetos en pixels.

```
#!/usr/bin/python
# ruler1.py

import wx


RW = 701 # ruler widht
RM = 10  # ruler margin
RH = 60  # ruler height


class Ruler1(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(RW + 2*RM, 60),
            style=wx.FRAME_NO_TASKBAR | wx.NO_BORDER | wx.STAY_ON_TOP)
        self.font = wx.Font(7, wx.FONTFAMILY_DEFAULT, wx.FONTSTYLE_NORMAL,
            wx.FONTWEIGHT_BOLD, False, 'Courier 10 Pitch')

        self.Bind(wx.EVT_PAINT, self.OnPaint)
        self.Bind(wx.EVT_LEFT_DOWN, self.OnLeftDown)
        self.Bind(wx.EVT_RIGHT_DOWN, self.OnRightDown)
        self.Bind(wx.EVT_MOTION, self.OnMouseMove)

        self.Centre()
        self.Show(True)

    def OnPaint(self, event):
        dc = wx.PaintDC(self)
```

```
        brush = wx.BrushFromBitmap(wx.Bitmap('granite.png'))
        dc.SetBrush(brush)
        dc.DrawRectangle(0, 0, RW+2*RM, RH)
        dc.SetFont(self.font)


        dc.SetPen(wx.Pen('#F8FF25'))
        dc.SetTextForeground('#F8FF25')


        for i in range(RW):
            if not (i % 100):
                dc.DrawLine(i+RM, 0, i+RM, 10)
                w, h = dc.GetTextExtent(str(i))
                dc.DrawText(str(i), i+RM-w/2, 11)
            elif not (i % 20):
                dc.DrawLine(i+RM, 0, i+RM, 8)
            elif not (i % 2): dc.DrawLine(i+RM, 0, i+RM, 4)

    def OnLeftDown(self, event):
        pos = event.GetPosition()
        x, y = self.ClientToScreen(event.GetPosition())
        ox, oy = self.GetPosition()
        dx = x - ox
        dy = y - oy
        self.delta = ((dx, dy))

    def OnMouseMove(self, event):
        if event.Dragging() and event.LeftIsDown():
            x, y = self.ClientToScreen(event.GetPosition())
            fp = (x - self.delta[0], y - self.delta[1])
            self.Move(fp)

    def OnRightDown(self, event):
        self.Close()

app = wx.App()
Ruler1(None, -1, '')
app.MainLoop()
```

En este ejemplo creamos una regla. Esta regla podrá medir objetos en la pantalla en pixels. Dejamos el modo de mapeo por omisión, el cual es *wx.MM_TEXT*. Como ya hemos mencionado, este modo tiene las mismas unidades lógicas y de dispositivo. En nuestro caso, pixels.

```
 wx.Frame.__init__(self, parent, id, title, size=(RW + 2*RM, 60),
style=wx.FRAME_NO_TASKBAR |
        wx.NO_BORDER | wx.STAY_ON_TOP)
```

Tenemos creados una ventana sin bordes. La regla tiene 721 px de ancho. La regla es RW + 2*RM = 701 + 20 = 721. La regla muestra los números 700. 0 ... 700 es 701 pixels. Una regla tiene un margen sobre ambos lados, 2*10 es 20 pixels. Juntos hacen 721 pixels.

```
 brush = wx.BrushFromBitmap(wx.Bitmap('granite.png'))
 dc.SetBrush(brush)
 dc.DrawRectangle(0, 0, RW+2*RM, RH)
```

Aquí dibujamos un patrón personalizado sobre la ventana. Uso un patrón predefinido disponible en el GIMP. Éste es llamado granite.

```
w, h = dc.GetTextExtent(str(i))
dc.DrawText(str(i), i+RM-w/2, 11)
```

Estas líneas aseguran, que alineamos el texto correctamente. El método *GetTextExtent()* retorna el ancho y el alto del texto.

No tenemos que hacer un borde alrededor de nuestras ventanas. Así debemos manejar moviendo manualmente con código adicional. Los métodos *OnLeftDown()* y el *OnMouseMove()* nos habilitan para mover la regla. (TODO:link to dragging.)



Figura: ejemplo primer regla

Ejemplos prácticos

Usted podría preguntarse, ¿Por qué necesitamos todos estas líneas, plumas y gradientes? Qué es lo bueno? Los siguientes guiones podrá llevar a algunos ejemplos prácticos. Podríamos utilizar, para aprender con la práctica.

Gráficos

Creando gráficos es un excelente ejemplo de utilización de funciones de dibujo gdi. Los gráficos no son componentes GUI. Los kits de herramientas gui no provee gráficos como parte de la biblioteca. Una es el kit de herramientas wxWidgets (y así el wxPython). Pero estos gráficos son muy simples y no pueden ser usados en aplicaciones reales. Un desarrollador tiene usualmente dos opciones. Para crear sus propias biblioteca de gráficos o usar una biblioteca de terceras partes.

En el siguiente ejemplo creamos un gráfico con una línea simple. No nos demoraremos en todos los detalles. Seguí el ejemplo intencionalmente simple. mucho de estas cosas permanecen sin hacerse. Pero usted puede comprender la idea y seguirla.

```
#!/usr/bin/python
# linechart.py

import wx

data = ((10, 9), (20, 22), (30, 21), (40, 30), (50, 41),
(60, 53), (70, 45), (80, 20), (90, 19), (100, 22),
(110, 42), (120, 62), (130, 43), (140, 71), (150, 89),
(160, 65), (170, 126), (180, 187), (190, 128), (200, 125),
(210, 150), (220, 129), (230, 133), (240, 134), (250, 165),
(260, 132), (270, 130), (280, 159), (290, 163), (300, 94))

years = ('2003', '2004', '2005')


class LineChart(wx.Panel):
    def __init__(self, parent):
        wx.Panel.__init__(self, parent)
        self.SetBackgroundColour('WHITE')
```

```
        self.Bind(wx.EVT_PAINT, self.OnPaint)

    def OnPaint(self, event):
        dc = wx.PaintDC(self)
        dc.SetDeviceOrigin(40, 240)
        dc.SetAxisOrientation(True, True)
        dc.SetPen(wx.Pen('WHITE'))
        dc.DrawRectangle(1, 1, 300, 200)
        self.DrawAxis(dc)
        self.DrawGrid(dc)
        self.DrawTitle(dc)
        self.DrawData(dc)

    def DrawAxis(self, dc):
        dc.SetPen(wx.Pen('#0AB1FF'))
        font =  dc.GetFont()
        font.SetPointSize(8)
        dc.SetFont(font)
        dc.DrawLine(1, 1, 300, 1)
        dc.DrawLine(1, 1, 1, 201)

        for i in range(20, 220, 20):
            dc.DrawText(str(i), -30, i+5)
            dc.DrawLine(2, i, -5, i)

        for i in range(100, 300, 100):
            dc.DrawLine(i, 2, i, -5)

        for i in range(3):
            dc.DrawText(years[i], i*100-13, -10)



    def DrawGrid(self, dc):
        dc.SetPen(wx.Pen('#d5d5d5'))

        for i in range(20, 220, 20):
            dc.DrawLine(2, i, 300, i)

        for i in range(100, 300, 100):
            dc.DrawLine(i, 2, i, 200)

    def DrawTitle(self, dc):
        font =  dc.GetFont()
        font.SetWeight(wx.FONTWEIGHT_BOLD)
        dc.SetFont(font)
        dc.DrawText('Historical Prices', 90, 235)


    def DrawData(self, dc):
        dc.SetPen(wx.Pen('#0ab1ff'))
        for i in range(10, 310, 10):
            dc.DrawSpline(data)
```

```
class LineChartExample(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(390, 300))

        panel = wx.Panel(self, -1)
        panel.SetBackgroundColour('WHITE')

        hbox = wx.BoxSizer(wx.HORIZONTAL)
        linechart = LineChart(panel)
        hbox.Add(linechart, 1, wx.EXPAND | wx.ALL, 15)
        panel.SetSizer(hbox)

        self.Centre()
        self.Show(True)


app = wx.App()
LineChartExample(None, -1, 'A line chart')
app.MainLoop()
```

```
dc.SetDeviceOrigin(40, 240)
dc.SetAxisOrientation(True, True)
```

Por omisión el sistema de coordenadas en wxPython inicia en el punto [0, 0]. El punto de comienzo está localizada en la esquina superior izquierda del área del cliente. La orientación del valor x es desde la izquierda a la derecha y la orientación del valor y es de arriba abajo. Los valores pueden ser solamente positivos. Este sistema es usado en todos los kits de herramientas GUI. (de todos los que soy conciente.)

Para gráficos usamos sistema cartesiano de coordenadas. En el sistema cartesiano, podemos tener valores tanto positivos y negativos. La orientación del valor x es desde la izquierda a la derecha y la orientación del valor y es desde el fondo hacia arriba. El origen es usualmente en el medio. Pero éste no es obligatorio.


```
dc.SetDeviceOrigin(40, 240)
dc.SetAxisOrientation(True, True)
```


El método *SetDeviceOrigin()* mueve el origen a un nuevo punto sobre el área cliente. Éste es llamado **traslado lineal**. Entonces cambiamos la orientación axis con el método *SetAxisOrientation()*.

```
SetAxisOrientation(bool xLeftRight, bool yBottomUp)
```

El método signature es se explica por si solo. Podemos poner valores true o false a estos dos parámetros.

```
self.DrawAxis(dc)
self.DrawGrid(dc)
self.DrawTitle(dc)
self.DrawData(dc)
```

Separamos la construction del gráfico dentro de cuatro métodos. El primer podrá dibujar el eje de axis, el segundo podrá dibujar la grilla, el tercero el título y el último uno podrá dibujar los datos.

```
for i in range(3):
    dc.DrawText(years[i], i*100-13, -10)
```

Porque de la simplicidad del guión, aquí son algunos números mágicos. En realidad, deberíamos tener que calcularlos. En el código previo de ejemplo, dibujamos los años a lo largo del eje axis x. Restamos 13 px desde el valor x. Éste se hace para centrar los años sobre las líneas verticales. Éste trabaja sobre mi versión de linux. Podría no trabajar correctamente sobre otras plataformas de SO. Ésto podría no trabajar incluso sobre versiones de linux con diferentes temas. Usted podrá jugar un poquito con este ejemplo. Ajustandolo para adaptarse bajo a diferentes circunstancias no es ciencia de cohetes. Normalmente, necesitamos calcular el ancho del gráfico, el ancho del texto y centrar el texto manualmente.

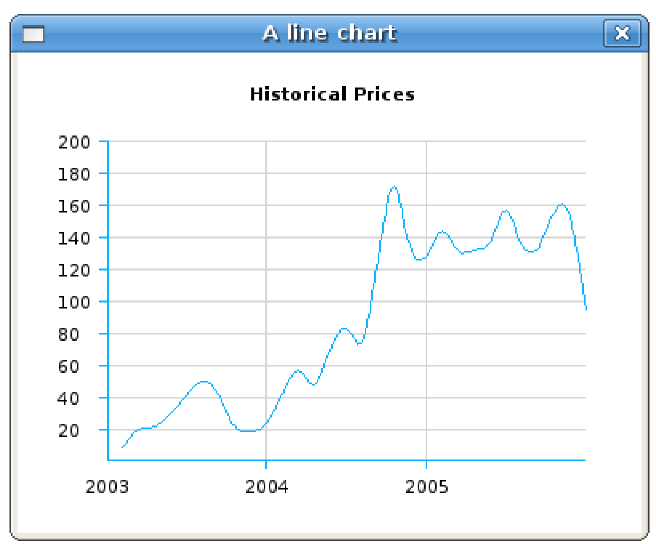


Figura: un gráfico de líneas

Nota

Nota es un guión pequeño que muestra varios características interesantes del GDI. Veremos, como podemos crear un personalizado en forma de ventana. Estas son pequeñas aplicaciones que son usadas para hacer notas visibles. Aquellos trabajan como recordatorios para personas, que trabajan muchocon computadoras. (p.e. nosotros).

```
#!/usr/bin/python
# note.py

import wx

class Note(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title,
                        style=wx.FRAME_SHAPED |
                        wx.SIMPLE_BORDER |
                        wx.FRAME_NO_TASKBAR)

        self.font = wx.Font(11, wx.FONTFAMILY_DEFAULT, wx.FONTSTYLE_NORMAL,
                wx.FONTWEIGHT_BOLD, False, 'Comic Sans MS')
        self.bitmap = wx.Bitmap('note.png', wx.BITMAP_TYPE_PNG)
        self.cross = wx.Bitmap('cross.png', wx.BITMAP_TYPE_PNG)

        w = self.bitmap.GetWidth()
        h = self.bitmap.GetHeight()
        self.SetClientSize((w, h))

        if wx.Platform == '__WXGTK__':
```

```
            self.Bind(wx.EVT_WINDOW_CREATE, self.SetNoteShape)
        else: self.SetNoteShape()

        self.Bind(wx.EVT_PAINT, self.OnPaint)
        self.Bind(wx.EVT_LEFT_DOWN, self.OnLeftDown)
        self.Bind(wx.EVT_MOTION, self.OnMouseMove)

        self.bitmapRegion = wx.RegionFromBitmap(self.bitmap)
        self.crossRegion = wx.RegionFromBitmap(self.cross)

        self.bitmapRegion.IntersectRegion(self.crossRegion)
        self.bitmapRegion.Offset(170, 10)

        dc = wx.ClientDC(self)
        dc.DrawBitmap(self.bitmap, 0, 0, True)
        self.PositionTopRight()
        self.Show(True)

    def PositionTopRight(self):
        disx, disy = wx.GetDisplaySize()
        x, y = self.GetSize()
        self.Move((disx-x, 0))

    def SetNoteShape(self, *event):
        region = wx.RegionFromBitmap(self.bitmap)
        self.SetShape(region)

    def OnLeftDown(self, event):
        pos = event.GetPosition()
        if self.bitmapRegion.ContainsPoint(pos):
            self.Close()
        x, y = self.ClientToScreen(event.GetPosition())
        ox, oy = self.GetPosition()
        dx = x - ox
        dy = y - oy
        self.delta = ((dx, dy))


    def OnMouseMove(self, event):
        if event.Dragging() and event.LeftIsDown():
            x, y = self.ClientToScreen(event.GetPosition())
            fp = (x - self.delta[0], y - self.delta[1])
            self.Move(fp)

    def OnPaint(self, event):
        dc = wx.PaintDC(self)
        dc.SetFont(self.font)
        dc.SetTextForeground('WHITE')

        dc.DrawBitmap(self.bitmap, 0, 0, True)
        dc.DrawBitmap(self.cross, 170, 10, True)
        dc.DrawText('- Go shopping', 20, 20)
        dc.DrawText('- Make a phone call', 20, 50)
        dc.DrawText('- Write an email', 20, 80)
```

```
app = wx.App()
Note(None, -1, '')
app.MainLoop()
```

La idea detras crear una en forma de ventana es simple. La mayoría de aplicaciones son rectangulares. Éstas comparten gran cantidad de similitudes. Éstas tienen menus, Barra de Herramientas, títulos etc. Esto sería aburrido. Algunos desarrolladores crean más aplicaciones de fantasía. Podemos hacer nuestras aplicaciones más atractivas pero usando imágenes. La idea es el siguiente. Creamos un marco sin borde. Podemos dibujar una imagen personalizada sobre el marco durante el evento pintar.

```
wx.Frame.__init__(self, parent, id, title,
                style=wx.FRAME_SHAPED |
                wx.SIMPLE_BORDER |
                wx.FRAME_NO_TASKBAR)
```

En orden para crear un personalizado en forma de aplicaciones, debemos fijar las opciones de esilo necesarias. El *wx.FRAME_SHAPED* permite para crear una en forma de ventana. El *wx.SIMPLE_BORDER* remueve el espesor del borde. El *wx.FRAME_NO_TASKBAR* previene que la aplicación aparezca sobre la barra de tareas.

```
self.font = wx.Font(11, wx.FONTFAMILY_DEFAULT, wx.FONTSTYLE_NORMAL,
               wx.FONTWEIGHT_BOLD, False, 'Comic Sans MS')
```

Estaba buscando una buena fuente para la nota de ejemplo. Finalmente elegí Comic Sans MS. Éste es un tipo de letra propietario. Los usuarios Linux deben instalar el paquete msttcorefonts. Si no tenemos instalado el tipo de letra, el sistema elige otro. Usualmente no es de nuestro gusto.

```
self.bitmap = wx.Bitmap('note.png', wx.BITMAP_TYPE_PNG)
self.cross = wx.Bitmap('cross.png', wx.BITMAP_TYPE_PNG)
```

Tengo creado dos bitmaps. El primero es un rectángulo redondeado. Con un tipo de relleno naranja. Esoty usando **Inkscape** vector illustrator para crearlo. El segundo es una pequeña cruz. Esto es usado para cerrar las aplicaciones. Para esto he usado el editor de imagenes **Gimp**.

```
w = self.bitmap.GetWidth()
h = self.bitmap.GetHeight()
self.SetClientSize((w, h))
```

Vamos a dibujar un mapa de bits sobre el marco. A fin de cubrir todo el marco, damos forma y tamaño al mapa de bits. Entonces fijamos el sitio del marcos al tamaño del bitmap.

```
if wx.Platform == '__WXGTK__':
    self.Bind(wx.EVT_WINDOW_CREATE, self.SetNoteShape)
else: self.SetNoteShape()
```

Éste es código dependiente de algunas plataformas. Los desarrolladores Linux deben llamar al método *SetNoteShape()* inmediatamente después del evento *wx.WindowCreateEvent*.

```
dc = wx.ClientDC(self)
dc.DrawBitmap(self.bitmap, 0, 0, True)
```

Estas líneas no son necesarias, porque un evento paint es generado durante la creación de las aplicaciones. Pero creemos, esto hace el ejemplo más suave. Digo que, porque ésto qué tengo que aprenderlo desde el otros.

```
def SetNoteShape(self, *event):
    region = wx.RegionFromBitmap(self.bitmap)
```

```
    self.SetShape(region)
```

Aquí fijamos la figura del marco a la del bitmap. Los pixels fuera de la imagen se convierten en transparentes.

Si removemos un borde desde el marco, no podemos mover la ventana. Los métodos *OnLeftDown()* y el *OnMouseMove()* permiten al usuario mover la ventana haciendo clic en el área cliente del marco y arrastrarla.

```
dc.DrawBitmap(self.bitmap, 0, 0, True)
dc.DrawBitmap(self.cross, 170, 10, True)
dc.DrawText('- Go shopping', 20, 20)
dc.DrawText('- Make a phone call', 20, 50)
dc.DrawText('- Write an email', 20, 80)
```

Dentro del método *OnPaint()* dibujamos dos bitmaps y tres textos.

Finalmente podremos hablar acerca de como cerramos el guión Nota.

```
self.bitmapRegion = wx.RegionFromBitmap(self.bitmap)
self.crossRegion = wx.RegionFromBitmap(self.cross)

self.bitmapRegion.IntersectRegion(self.crossRegion)
self.bitmapRegion.Offset(170, 10)
...
pos = event.GetPosition()
if self.bitmapRegion.ContainsPoint(pos):
    self.Close()
```

Creamos dos regiones desde dos bitmaps. Cruzamos estas dos regiones. De esta manera tomamos todos los pixels que comparten ambos bitmaps. Finalmente movemos la región al punto, donde dibujamos el bitmap cruz. Usamos el método *Offset()*. Por omisión la región arranca en el punto [0, 0].

Dentro del método *OnLeftDown()* comprobamos si hemos cliqueadod dentro de la región. Si es verdad, cerramos el guión.



Figura: Nota

En este capítulo hemos trabajado con el GDI.

Consejos y Trucos

En esta sección mostraremos varios consejos interesantes en wxPython. Aquí veremos ejemplos, que no podían ser puestos en otro lugar.

Botón Interactivo

Cuando entramos el área del componente botón con el puntero del ratón, el evento wx.EVT_ENTER_WINDOW es generado. Del mismo modo, el evento wx.EVT_LEAVE_WINDOW es generado, cuando salimos del área del componente. Vinculamos dos métodos a estos eventos.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

'''
ZetCode wxPython tutorial

Este ejemplo muestra un botón interactivo.

author: Jan Bodnar
website: www.zetcode.com
last modified: September 2011
'''

import wx
from wx.lib.buttons import GenButton

class Example(wx.Frame):

    def __init__(self, *args, **kwargs):
        super(Example, self).__init__(*args, **kwargs)

        self.InitUI()

    def InitUI(self):

        panel = wx.Panel(self)

        btn = GenButton(panel, label='botón',
            pos=(100, 100))
        btn.SetBezelWidth(1)
        btn.SetBackgroundColour('DARKGREY')

        wx.EVT_ENTER_WINDOW(btn, self.OnEnter)
        wx.EVT_LEAVE_WINDOW(btn, self.OnLeave)

        self.SetSize((300, 200))
        self.SetTitle('Interactive botón')
        self.Centre()
        self.Show(True)

    def OnEnter(self, e):

        btn = e.GetEventObject()
        btn.SetBackgroundColour('GREY79')
        btn.Refresh()

    def OnLeave(self, e):

        btn = e.GetEventObject()
        btn.SetBackgroundColour('DARKGREY')
        btn.Refresh()
```

```
def main():

    ex = wx.App()
    Example(None)
    ex.MainLoop()


if __name__ == '__main__':
    main()
```

Estamos usando a GenButton en lugar del wx.button basico.

```
from wx.lib.buttons import GenButton
```

El GenButton está localizado en el módulo wx.lib.buttons.

```
btn.SetBezelWidth(1)
```

El método SetBezelWidth() crea algunos efectos 3D sobre el botón.

```
def OnEnter(self, e):

    btn = e.GetEventObject()
    btn.SetBackgroundColour('GREY79')
    btn.Refresh()
```

En reacción al wx.EVT_ENTER_WINDOW, cambiamos el color de fondo del botón.

Isabelle

Cuando un error ocurre en una Aplicación, un cuadro de diálogo de error usualmente aparece. Este podría ser molesto. He notado una mejor solución en un sistema SAP. Cuando un usuario entra un comando invalido, la barra de estado se pone roja y un mensaje de error es mostrado sobre la barra de estado. El color rojo atrapa al ojo y el usuario pueden fácilmente leer el mensaje de error. El código siguiente imita esta situación.

```
#!/usr/bin/python

# Isabelle

import wx

ID_TIMER = 1
ID_EXIT  = 2
ID_ABOUT = 3
ID_button = 4

class Isabelle(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title)

        self.timer = wx.Timer(self, ID_TIMER)
        self.blick = 0

        file = wx.Menu()
        file.Append(ID_EXIT, '&Quit\tCtrl+Q', 'QuÉste esabelle')
```

```
        help = wx.Menu()
        help.Append(ID_ABOUT, '&About', 'O Programe')


        menubar = wx.MenuBar()
        menubar.Append(file, '&File')
        menubar.Append(help, '&Help')
        self.SetMenuBar(menubar)

        toolbar = wx.ToolBar(self, -1)
        self.tc = wx.TextCtrl(toolbar, -1, size=(100, -1))
        btn = wx.button(toolbar, ID_botón, 'Ok', size=(40, 28))

        toolbar.AddControl(self.tc)
        toolbar.AddSeparator()
        toolbar.AddControl(btn)
        toolbar.Realize()
        self.SetToolBar(toolbar)

        self.Bind(wx.EVT_BUTTON, self.OnLaunchCommandOk, id=ID_botón)
        self.Bind(wx.EVT_MENU, self.OnAbout, id=ID_ABOUT)
        self.Bind(wx.EVT_MENU, self.OnExit, id=ID_EXIT)
        self.Bind(wx.EVT_TIMER, self.OnTimer, id=ID_TIMER)

        self.panel = wx.Panel(self, -1, (0, 0), (500 , 300))
        self.panel.SetBackgroundColour('GRAY')
        self.sizer=wx.BoxSizer(wx.VERTICAL)
        self.sizer.Add(self.panel, 1, wx.EXPAND)
        self.SetSizer(self.sizer)
        self.statusbar = self.CreateStatusBar()
        self.statusbar.SetStatusText('Welcome to Isabelle')
        self.Centre()
        self.Show(True)

    def OnExit(self, event):
        dlg = wx.MessageDialog(self, 'Are you sure to quÉste esabelle?',
            'Please Confirm', wx.YES_NO | wx.NO_DEFAULT | wx.ICON_QUESTION)
        if dlg.ShowModal() == wx.ID_YES:
            self.Close(True)


    def OnAbout(self, event):
        dlg = wx.MessageDialog(self, 'Isabelle\t\n' '2004\t', 'About',
                 wx.OK | wx.ICON_INFORMATION)
        dlg.ShowModal()
        dlg.Destroy()


    def OnLaunchCommandOk(self, event):
        input = self.tc.GetValue()
        if input == '/bye':
            self.OnExit(self)
        elif input == '/about':
            self.OnAbout(self)
        elif input == '/bell':
```

```
            wx.Bell()
        else:
            self.statusbar.SetBackgroundColour('RED')
            self.statusbar.SetStatusText('Unknown Command')
            self.statusbar.Refresh()
            self.timer.Start(50)

        self.tc.Clear()

    def OnTimer(self, event):
        self.blick = self.blick + 1
        if self.blick == 25:
            self.statusbar.SetBackgroundColour('#E0E2EB')
            self.statusbar.Refresh()
            self.timer.Stop()
            self.blick = 0

app = wx.App()
Isabelle(None, -1, 'Isabelle')
app.MainLoop()
```

Ésta es un wx.TextCtrl sobre la barra de estado. Usted entra vuestros comandos. Tenemos definidos tres comandos. /bye, /about y /beep. Si usted se equivoca en cualquiera de ellos, la barra de estado se torna roja y muestra un error. Éste es hecho con la clase wx.Timer.

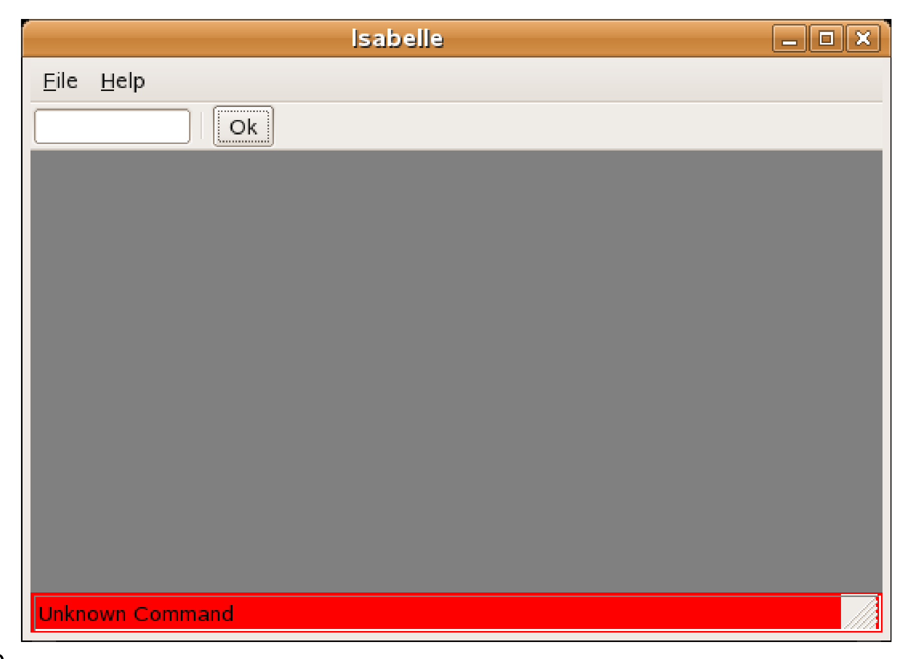


Figura: Isabelle

Marco de trabajo para deshacer/rehacer

Muchas aplicaciones tienen la habilidad de deshacer y rehacer las acciones del usuario. El siguiente ejemplo muestra como lo pueden realizar en wxPython.

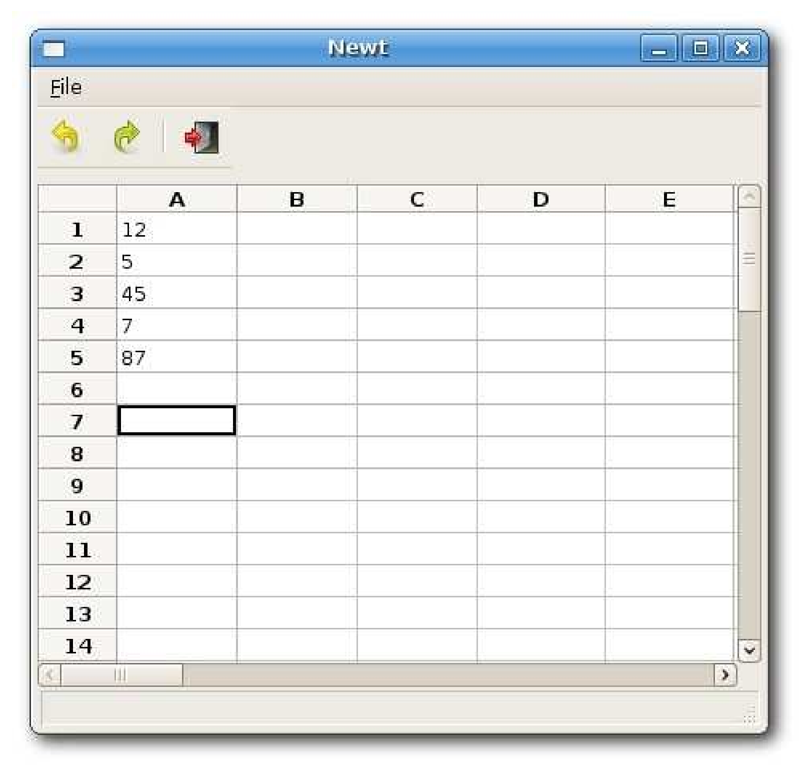


Figura: undoredo.py

```
#!/usr/bin/python
# undoredo.py

from wx.lib.sheet import *
```

```
import wx

stockUndo = []
stockRedo = []

ID_QUIT = 10
ID_UNDO = 11
ID_REDO = 12
ID_EXIT = 13

ID_COLSIZE = 80
ID_ROWSIZE = 20


class UndoText:
    def __init__(self, sheet, text1, text2, row, column):
        self.RedoText =  text2
        self.row = row
        self.col = column
        self.UndoText = text1
        self.sheet = sheet

    def undo(self):
        self.RedoText = self.sheet.GetCellValue(self.row, self.col)
        if self.UndoText ==  None:
            self.sheetSetCellValue('')
        else: self.sheet.SetCellValue(self.row, self.col, self.UndoText)

    def redo(self):
        if self.RedoText == None:
            self.sheet.SetCellValue('')
        else: self.sheet.SetCellValue(self.row, self.col, self.RedoText)

class UndoColSize:
    def __init__(self, sheet, position, size):
        self.sheet = sheet
        self.pos = position
        self.RedoSize = size
        self.UndoSize = ID_COLSIZE

    def undo(self):
        self.RedoSize = self.sheet.GetColSize(self.pos)
        self.sheet.SetColSize(self.pos, self.UndoSize)
        self.sheet.ForceRefresh()

    def redo(self):
        self.UndoSize = ID_COLSIZE
        self.sheet.SetColSize(self.pos, self.RedoSize)
        self.sheet.ForceRefresh()

class UndoRowSize:
    def __init__(self, sheet, position, size):
        self.sheet = sheet
        self.pos = position
        self.RedoSize = size
        self.UndoSize = ID_ROWSIZE
```

```
    def undo(self):
        self.RedoSize = self.sheet.GetRowSize(self.pos)
        self.sheet.SetRowSize(self.pos, self.UndoSize)
        self.sheet.ForceRefresh()

    def redo(self):
        self.UndoSize = ID_ROWSIZE
        self.sheet.SetRowSize(self.pos, self.RedoSize)
        self.sheet.ForceRefresh()

class MySheet(CSheet):
    instance = 0
    def __init__(self, parent):
        CSheet.__init__(self, parent)
        self.SetRowLabelAlignment(wx.ALIGN_CENTRE, wx.ALIGN_CENTRE)
        self.text = ''

    def OnCellChange(self, event):
        toolbar = self.GetParent().toolbar
        if (toolbar.GetToolEnabled(ID_UNDO) == False):
                toolbar.EnableTool(ID_UNDO, True)
        r = event.GetRow()
        c = event.GetCol()
        text = self.GetCellValue(r, c)
        # self.text - text antes del change
        # text - text Después de change
        deshacer = UndoText(self, self.text, text, r, c)
        stockUndo.append(undo)

        if stockRedo:
            # this sería surprising, pero Esto es una estándares
comportamiento
            # en all se hoja de cálculos
            del stockRedo[:]
            toolbar.EnableTool(ID_REDO, False)

    def OnColSize(self, event):
        toolbar = self.GetParent().toolbar

        if (toolbar.GetToolEnabled(ID_UNDO) == False):
                toolbar.EnableTool(ID_UNDO, True)

        pos =  event.GetRowOrCol()
        size = self.GetColSize(pos)
        deshacer = UndoColSize(self, pos, size)
        stockUndo.append(undo)

        if stockRedo:
            del stockRedo[:]
            toolbar.EnableTool(ID_REDO, False)

    def OnRowSize(self, event):
        toolbar = self.GetParent().toolbar
        if (toolbar.GetToolEnabled(ID_UNDO) == False):
                toolbar.EnableTool(ID_UNDO, True)
```

```
        pos =  event.GetRowOrCol()
        size = self.GetRowSize(pos)
        deshacer = UndoRowSize(self, pos, size)

        stockUndo.append(undo)
        if stockRedo:
            del stockRedo[:]
            toolbar.EnableTool(ID_REDO, False)

class Newt(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self, parent, -1, title, size=(550, 500))

        box = wx.BoxSizer(wx.VERTICAL)
        menubar = wx.MenuBar()
        menu = wx.Menu()
        quit = wx.MenuItem(menu, ID_QUIT, '&Quit\tCtrl+Q', 'Quits Newt')
        quit.SetBitmap(wx.Bitmap('icons/exit16.png'))
        menu.AppendItem(quit)
        menuBar.Append(menu, '&File')
        self.Bind(wx.EVT_MENU, self.OnQuitNewt, id=ID_QUIT)
        self.SetMenuBar(menuBar)


        self.toolbar = wx.ToolBar(self, id=-1, style=wx.TB_HORIZONTAL |
wx.NO_BORDER |
                                        wx.TB_FLAT | wx.TB_TEXT)
        self.toolbar.AddSimpleTool(ID_UNDO, wx.Bitmap('icons/undo.png'),
              'Undo', '')
        self.toolbar.AddSimpleTool(ID_REDO, wx.Bitmap('icons/redo.png'),
              'Redo', '')
        self.toolbar.EnableTool(ID_UNDO, False)

        self.toolbar.EnableTool(ID_REDO, False)
        self.toolbar.AddSeparator()
        self.toolbar.AddSimpleTool(ID_EXIT, wx.Bitmap('icons/exit.png'),
              'Quit', '')
        self.toolbar.Realize()
        self.toolbar.Bind(wx.EVT_TOOL, self.OnUndo, id=ID_UNDO)
        self.toolbar.Bind(wx.EVT_TOOL, self.OnRedo, id=ID_REDO)
        self.toolbar.Bind(wx.EVT_TOOL, self.OnQuitNewt, id=ID_EXIT)

        box.Add(self.toolbar, border=5)
        box.Add((5,10), 0)

        self.SetSizer(box)
        self.sheet1 = MySheet(self)
        self.sheet1.SetNumberRows(55)
        self.sheet1.SetNumberCols(25)

        for i in range(self.sheet1.GetNumberRows()):
            self.sheet1.SetRowSize(i, ID_ROWSIZE)

        self.sheet1.SetFocus()
        box.Add(self.sheet1, 1, wx.EXPAND)
```

```
        self.CreateStatusBar()
        self.Centre()
        self.Show(True)

    def OnUndo(self, event):
        if len(stockUndo) == 0:
            return

        a = stockUndo.pop()
        if len(stockUndo) == 0:
            self.toolbar.EnableTool(ID_UNDO, False)

        a.undo()
        stockRedo.append(a)
        self.toolbar.EnableTool(ID_REDO, True)

    def OnRedo(self, event):
        if len(stockRedo) == 0:
            return

        a = stockRedo.pop()
        if len(stockRedo) == 0:
            self.toolbar.EnableTool(ID_REDO, False)

        a.redo()
        stockUndo.append(a)

        self.toolbar.EnableTool(ID_UNDO, True)

    def OnQuitNewt(self, event):
        self.Close(True)

app = wx.App()
Newt(None, -1, 'Newt')
app.MainLoop()
```

```
stockUndo = []
stockRedo = []
```

Estos son dos objetos lista. stockUndo es una lista que retiene todos los cambios, que podemos deshacer. stockRedo mantiene todos los cambios, que pueden ser rehechos. Los cambios son instanciados dentro de un objeto UndoText. Este objeto tiene dos métodos. Undo y redo.

```
class MySheet(CSheet):
    def __init__(self, parent):
        CSheet.__init__(self, parent)
```

Nuestro ejemplo hereda desde la clase CSheet. Esto es una componente grilla con alguna lógica adicional.

```
self.SetRowLabelAlignment(wx.ALIGN_CENTRE, wx.ALIGN_CENTRE)
```

Aquí centramos la etiquetas en filas. Por omisión, aquellos son aligned a la derecha.

```
r = event.GetRow()
c = event.GetCol()
```

```
text = self.GetCellValue(r, c)
# self.text - text antes del change
# text - text Después de change
deshacer = UndoText(self, self.text, text, r, c)
stockUndo.append(undo)
```

Cada vez que hacemos algunos cambios, un objeto UndoText es creado y anexado a la lista de stockUndo.

```
if stockRedo:
    # this sería surprising, pero Esto es una estándares comportamiento
    # en todos los se hoja de cálculos
    del stockRedo[:]
        toolbar.EnableTool(ID_REDO, False)
```

Sí, este comportamiento fue sorprendente para mi.No sabía que esto trabajaba ee esta manera, hasta que hice este ejemplo. Basicamente, si usted deshace algunos cambios y entonces comienzan tipeando de nuevo, todos los cambios rehacer son perdido. OpenOffice Calc trabaja de esta manera. Gnumeric también.

```
if len(stockUndo) == 0:
    self.toolbar.EnableTool(ID_UNDO, False)
...
self.toolbar.EnableTool(ID_REDO, True)
```

Los botones deshacer y rehacer son activado o desactivado en consecuencia. Si no hay que desahacer, el botón deshacer está deshabilitado.

```
a = stockUndo.pop()
if len(stockUndo) == 0:
    self.toolbar.EnableTool(ID_UNDO, False)

a.undo()
stockRedo.append(a)
```

Si hacemos clic en undo, hacemos aparecer un objeto UndoText desde la lista stockUndo. Llamar al método *undo()* y agregar el objeto a la lista stockRedo.

Configurando ajustes de aplicaciones

Muchas aplicaciones permite a los usuarios configurar sus ajustes. Los usuarios pueden cambiar tooltips de activos a no, cambiar tipos de letra, rutas de descarga de archivos por omisión etc. En su mayoría aquellos tienen una opción de menu llamada preferencias. Las configuraciones de las aplicaciones son guardadas al disco duro, de modo que usuarios no tienen que cambiar los ajustes cada vez que las aplicaciones arrancan.

En wxPython tenemos la clase *wx.Config* para hacer nuestra trabajo.

Sobre Linux, los ajustes son almacenados en un simple archivo oculto. Este archivo está localizada en el directorio raiz del usuario (home) por omisión. La ubicación del archivo de configuración puede ser cambiada. El nombre del archivo es especificado en el constructor de la clase *wx.Config*. En el siguiente código de ejemplo, podemos configurar el tamaño de la ventana. Si esta el archivo de configuración, el alto y el ancho de la ventana es fijado al valor por omisión 250 px. Podemos fijar estos valores a un rango de 200 a 500px. Después de salvamos nuestros valores y reiniciamos las aplicaciones, el tamaño de la ventana es fijada a nuestros valores preferidos.

```
#!/usr/bin/python
```

```
# myconfig.py

import wx

class MyConfig(wx.Frame):
    def __init__(self, parent, id, title):
        self.cfg = wx.Config('myconfig')
        if self.cfg.Exists('width'):
            w, h = self.cfg.ReadInt('width'), self.cfg.ReadInt('height')
        else:
            (w, h) = (250, 250)
        wx.Frame.__init__(self, parent, id, title, size=(w, h))

        wx.StaticText(self, -1, 'Width:', (20, 20))
        wx.StaticText(self, -1, 'Height:', (20, 70))
        self.sc1 = wx.SpinCtrl(self, -1, str(w), (80, 15), (60, -1),
min=200, max=500)
        self.sc2 = wx.SpinCtrl(self, -1, str(h), (80, 65), (60, -1),
min=200, max=500)
        wx.button(self, 1, 'Save', (20, 120))

        self.Bind(wx.EVT_BUTTON, self.OnSave, id=1)
        self.statusbar = self.CreateStatusBar()
        self.Centre()
        self.Show(True)

    def OnSave(self, event):
        self.cfg.WriteInt("width", self.sc1.GetValue())
        self.cfg.WriteInt("height", self.sc2.GetValue())
        self.statusbar.SetStatusText('Configuration saved, %s ' % wx.Now())


app = wx.App()
MyConfig(None, -1, 'myconfig.py')
app.MainLoop()
```

Aquí tenemos los contenidos de un archivo de configuración para nuestros código de ejemplo. Éste consiste de las dos pares de clave, valor.

```
$ cat .myconfig
height=230
```

```
width=350
```



Figura: myconfig.py

Gestos con el ratón

Un gesto con el ratón es una forma de combinar movimientos del ratón de la computadora y clics los cuales el software reconoce como un comando específico. Podemos encontrar gesto con el ratón en aplicaciones tales como Firefox u Opera. Aquellos ayudan a los usuarios a ahorrar su tiempo mientras navegan en Internet. Los gestos con el ratón son creados con la clase wx.lib.gestures.MouseGestures en wxPython.

Gestos disponibles:

- L for left
- R for right
- U for up
- D for abajo
- 7 for northwest
- 9 for northeast
- 1 for southwest
- 3 for southeast

Si usted se maravilla por qué estos números fueron elegidos, tenga a la vista el teclado numérico. Los gestos con el ratón pueden ser combinados. De esta manera 'RDLU' es un gesto con el ratón se dispara, cuando hacemos un cuadrado con el puntero del ratón.

Posibles señales son:

- wx.MOUSE_BTN_LEFT
- wx.MOUSE_BTN_MIDDLE
- wx.MOUSE_BTN_RIGHT

```
#!/usr/bin/python

# mousegestures.py

import wx
import wx.lib.gestures as gest

class MyMouseGestures(wx.Frame):

    def __init__ (self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(300, 200))

        panel = wx.Panel(self, -1)
```

```
        mg = gest.MouseGestures(panel, True, wx.MOUSE_BTN_LEFT)
        mg.SetGesturePen(wx.Colour(255, 0, 0), 2)
        mg.SetGesturesVisible(True)
        mg.AddGesture('DR', self.OnDownRight)

        self.Centre()
        self.Show(True)

    def OnDownRight(self):
          self.Close()

app = wx.App()
MyMouseGestures(None, -1, 'mousegestures.py')
app.MainLoop()
```

En nuestro ejemplo, tenemos registrado un gesto con el ratón para un panel. El gesto del ratón es disparado, cuando el botón izquierdo es pulsado y vamos abajo y a la derecha con el cursor. Como en una letra 'L'. Nuestras gesto con el ratón cerrará las aplicaciones.

```
mg = gest.MouseGestures(panel, True, wx.MOUSE_BTN_LEFT)
```

Si buscamos para usar gestos con los ratones, tenemos que crear un objeto MouseGesture. El primer parámetro es una ventana, donde el gesto con el ratón es registrado. El segundo parámetro define un camiono para registrar el gesto. True es para automatico, False para manual. Manual no está completamente implementadas y estamos felices con el modo automático. El último parámetro define un botón del ratón, el cual podrá ser presionado cuando se disparen gestos. El botón pueden ser después cambiado con el método SetMouseButton().

```
mg.SetGesturePen(wx.Colour(255, 0, 0), 2)
```

Nuestros gestos podrán ser pintados como líneas rojas. Aquellos podrán ser de 2 pixels de ancho.

```
mg.SetGesturesVisible(True)
```

Fijamos this gesto visible con el método SetGesturesVisible().

```
 mg.AddGesture('DR', self.OnDownRight)
```

Registramos un gesto con el ratón con el método AddGesture(). El primer parámetro es el gesto. El segundo parámetro es el método disparado por el gesto.

En este capítulo, presentamos algunos consejos en wxPython.

wxPython Gripts

En esta sección mostraremos algunos guiones pequeños y completos. Estos guiones gráficos o "gripts" podrán demostrar varias áreas en programación.

Tom

Cada aplicaciones debería tener un buen nombre. Corto y fácilmente recordable. Por eso, tenemos

Tom. Un gript simple que envia un correo electrónico.



Figura: Tom

```
#!/usr/bin/python
# Tom

import wx
import smtplib

class Tom(wx.Dialog):
    def __init__(self, parent, id, title):
        wx.Dialog.__init__(self, parent, id, title, size=(400, 420))

        panel = wx.Panel(self, -1)
        vbox = wx.BoxSizer(wx.VERTICAL)
        hbox1 = wx.BoxSizer(wx.HORIZONTAL)
        hbox2 = wx.BoxSizer(wx.HORIZONTAL)
        hbox3 = wx.BoxSizer(wx.HORIZONTAL)

        st1 = wx.StaticText(panel, -1, 'From')
        st2 = wx.StaticText(panel, -1, 'To ')
        st3 = wx.StaticText(panel, -1, 'Subject')

        self.tc1 = wx.TextCtrl(panel, -1, size=(180, -1))
        self.tc2 = wx.TextCtrl(panel, -1, size=(180, -1))
        self.tc3 = wx.TextCtrl(panel, -1, size=(180, -1))

        self.write = wx.TextCtrl(panel, -1, style=wx.TE_MULTILINE)
        botón_send = wx.button(panel, 1, 'Send')

        hbox1.Add(st1, 0, wx.LEFT, 10)
```

```
        hbox1.Add(self.tc1, 0, wx.LEFT, 35)
        hbox2.Add(st2, 0, wx.LEFT, 10)
        hbox2.Add(self.tc2, 0, wx.LEFT, 50)
        hbox3.Add(st3, 0, wx.LEFT, 10)
        hbox3.Add(self.tc3, 0, wx.LEFT, 20)
        vbox.Add(hbox1, 0, wx.TOP, 10)
        vbox.Add(hbox2, 0, wx.TOP, 10)
        vbox.Add(hbox3, 0, wx.TOP, 10)
        vbox.Add(self.write, 1, wx.EXPAND | wx.TOP | wx.RIGHT | wx.LEFT, 15)
        vbox.Add(botón_send, 0, wx.ALIGN_CENTER | wx.TOP | wx.BOTTOM, 20)

        self.Bind(wx.EVT_BUTTON, self.OnSend, id=1)
        panel.SetSizer(vbox)

        self.Centre()
        self.ShowModal()
        self.Destroy()

    def OnSend(self, event):
        sender = self.tc1.GetValue()
        recipient = self.tc2.GetValue()
        subject = self.tc3.GetValue()
        text = self.write.GetValue()
        cabecera = 'From: %s\r\nTo: %s\r\nSubject: %s\r\n\r\n' % (sender, 
recipient, subject)
        mensaje = cabecera + text

        try:
            server = smtplib.SMTP('mail.chello.sk')
            server.sendmail(sender, recipient, mensaje)
            server.quit()
            dlg = wx.MessageDialog(self, 'Email was successfully sent', 
'Success',
                wx.OK | wx.ICON_INFORMATION)
            dlg.ShowModal()
            dlg.Destroy()

        except smtplib.SMTPException, error:
            dlg = wx.MessageDialog(self, 'Failed to send email', 'Error', 
wx.OK | wx.ICON_ERROR)
            dlg.ShowModal()
            dlg.Destroy()


app = wx.App()
Tom(None, -1, 'Tom')
app.MainLoop()
```

Para trabajar con emails necesitamos importar el módulo smtp. Este módulo es parte del lenguaje python.

```
import smtplib
```

Las opciones From, To y Subject deben ser separadas por retorno de carro y nueva línea como se muestra aquí. Este es extraño y requerido por la norma RFC 821. Así debemos seguirla.

```
header = 'From: %s\r\nTo: %s\r\nSubject: %s\r\n\r\n' % (sender, recipient,
subject)
```

A continuación creamos una conexión SMTP. Aquí usted especifica vuestros ajustes. Cada ISP le da a usted el nombre de los servidores pop y smtp. En mi caso, 'mail.chello.sk' es un nombre para ambos. Un correo es enviado al llamar el método sendmail(). Finalmente, salimos de la conexión con el método quit().

```
server = smtplib.SMTP('mail.chello.sk')
server.sendmail(sender, recipient, mensaje)
server.quit()
```

Editor

Este editor ejemplo es el más grande hasta ahora.

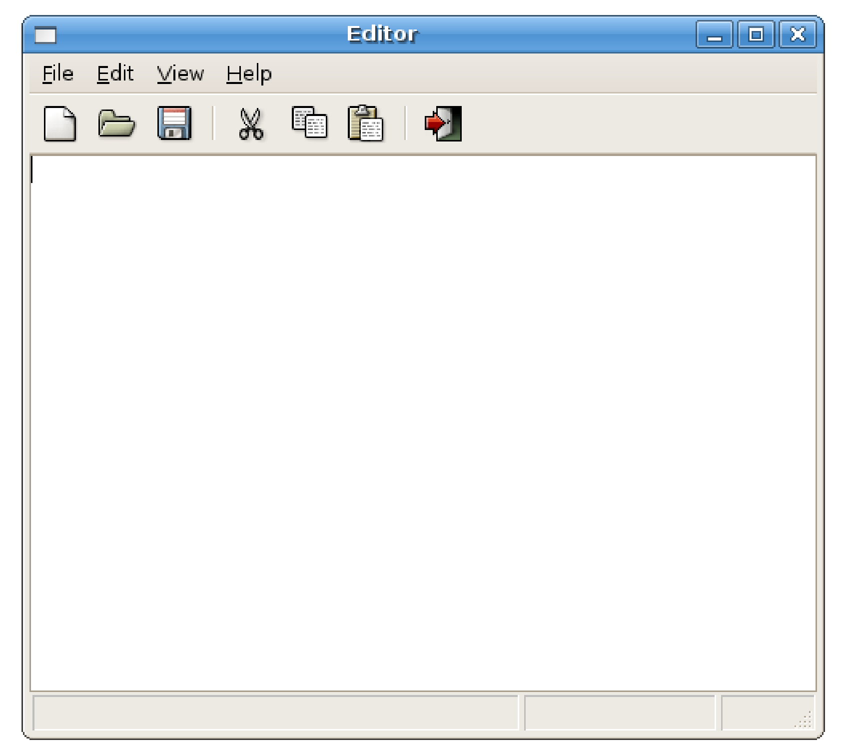


Figura: Editor

```
#!/usr/bin/python
# Editor

import wx
import os

class Editor(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(600, 500))

        # variables
        self.modify = False
        self.last_name_saved = ''
        self.replace = False
```

```
        # setting up menubar
        menubar = wx.MenuBar()

        file = wx.Menu()
        new = wx.MenuItem(file, 101, '&New\tCtrl+N', 'Creates a new
document')
        new.SetBitmap(wx.Bitmap('icons/stock_new-16.png'))
        file.AppendItem(new)

        open = wx.MenuItem(file, 102, '&Open\tCtrl+O', 'Open an existing
file')
        open.SetBitmap(wx.Bitmap('icons/stock_open-16.png'))
        file.AppendItem(open)
        file.AppendSeparator()

        save = wx.MenuItem(file, 103, '&Save\tCtrl+S', 'Save the file')
        save.SetBitmap(wx.Bitmap('icons/stock_save-16.png'))
        file.AppendItem(save)

        saveas = wx.MenuItem(file, 104, 'Save &As...\tShift+Ctrl+S',
                'Save the file with a different name')
        saveas.SetBitmap(wx.Bitmap('icons/stock_save_as-16.png'))
        file.AppendItem(saveas)
        file.AppendSeparator()

        quit = wx.MenuItem(file, 105, '&Quit\tCtrl+Q', 'Quit the
Application')
        quit.SetBitmap(wx.Bitmap('icons/stock_exit-16.png'))
        file.AppendItem(quit)

        edit = wx.Menu()
        cut = wx.MenuItem(edit, 106, '&Cut\tCtrl+X', 'Cut the Selection')
        cut.SetBitmap(wx.Bitmap('icons/stock_cut-16.png'))
        edit.AppendItem(cut)

        copy = wx.MenuItem(edit, 107, '&Copy\tCtrl+C', 'Copy the Selection')
        copy.SetBitmap(wx.Bitmap('icons/stock_copy-16.png'))
        edit.AppendItem(copy)

        paste = wx.MenuItem(edit, 108, '&Paste\tCtrl+V', 'Paste text from
clipboard')
        paste.SetBitmap(wx.Bitmap('icons/stock_paste-16.png'))
        edit.AppendItem(paste)

        delete = wx.MenuItem(edit, 109, '&Delete', 'Delete the selected
text')
        delete.SetBitmap(wx.Bitmap('icons/stock_delete-16.png',))

        edit.AppendItem(delete)
        edit.AppendSeparator()
        edit.Append(110, 'Select &All\tCtrl+A', 'Select the entire text')

        view = wx.Menu()
        view.Append(111, '&Statusbar', 'Show StatusBar')
```

```
        help = wx.Menu()
        about = wx.MenuItem(help, 112, '&About\tF1', 'About Editor')
        about.SetBitmap(wx.Bitmap('icons/stock_about-16.png'))
        help.AppendItem(about)

        menubar.Append(file, '&File')
        menubar.Append(edit, '&Edit')
        menubar.Append(view, '&View')
        menubar.Append(help, '&Help')
        self.SetMenuBar(menubar)

        self.Bind(wx.EVT_MENU, self.NewApplication, id=101)
        self.Bind(wx.EVT_MENU, self.OnOpenFile, id=102)
        self.Bind(wx.EVT_MENU, self.OnSaveFile, id=103)
        self.Bind(wx.EVT_MENU, self.OnSaveAsFile, id=104)
        self.Bind(wx.EVT_MENU, self.QuitApplication, id=105)
        self.Bind(wx.EVT_MENU, self.OnCut, id=106)
        self.Bind(wx.EVT_MENU, self.OnCopy, id=107)
        self.Bind(wx.EVT_MENU, self.OnPaste, id=108)
        self.Bind(wx.EVT_MENU, self.OnDelete, id=109)
        self.Bind(wx.EVT_MENU, self.OnSelectAll, id=110)
        self.Bind(wx.EVT_MENU, self.ToggleStatusBar, id=111)
        self.Bind(wx.EVT_MENU, self.OnAbout, id=112)

        # setting up toolbar
        self.toolbar = self.CreateToolBar( wx.TB_HORIZONTAL | wx.NO_BORDER |
wx.TB_FLAT
                | wx.TB_TEXT )
        self.toolbar.AddSimpleTool(801, wx.Bitmap('icons/stock_new.png'),
'New', '')
        self.toolbar.AddSimpleTool(802, wx.Bitmap('icons/stock_open.png'),
'Open', '')
        self.toolbar.AddSimpleTool(803, wx.Bitmap('icons/stock_save.png'),
'Save', '')
        self.toolbar.AddSeparator()

        self.toolbar.AddSimpleTool(804, wx.Bitmap('icons/stock_cut.png'),
'Cut', '')
        self.toolbar.AddSimpleTool(805, wx.Bitmap('icons/stock_copy.png'),
'Copy', '')
        self.toolbar.AddSimpleTool(806, wx.Bitmap('icons/stock_paste.png'),
'Paste', '')
        self.toolbar.AddSeparator()

        self.toolbar.AddSimpleTool(807, wx.Bitmap('icons/stock_exit.png'),
'Exit', '')
        self.toolbar.Realize()

        self.Bind(wx.EVT_TOOL, self.NewApplication, id=801)
        self.Bind(wx.EVT_TOOL, self.OnOpenFile, id=802)
        self.Bind(wx.EVT_TOOL, self.OnSaveFile, id=803)
        self.Bind(wx.EVT_TOOL, self.OnCut, id=804)
        self.Bind(wx.EVT_TOOL, self.OnCopy, id=805)
        self.Bind(wx.EVT_TOOL, self.OnPaste, id=806)
        self.Bind(wx.EVT_TOOL, self.QuitApplication, id=807)
```

```
        self.text = wx.TextCtrl(self, 1000, '', size=(-1, -1),
style=wx.TE_MULTILINE
                | wx.TE_PROCESS_ENTER)
        self.text.SetFocus()
        self.text.Bind(wx.EVT_TEXT, self.OnTextChanged, id=1000)
        self.text.Bind(wx.EVT_KEY_DOWN, self.OnKeyDown)

        self.Bind(wx.EVT_CLOSE, self.QuitApplication)

        self.StatusBar()

        self.Centre()
        self.Show(True)

    def NewApplication(self, event):
        editor = Editor(None, -1, 'Editor')
        editor.Centre()
        editor.Show()

    def OnOpenFile(self, event):
        file_name = os.path.basename(self.last_name_saved)
        if self.modify:
            dlg = wx.MessageDialog(self, 'Save changes?', '', wx.YES_NO |
wx.YES_DEFAULT |
         wx.CANCEL | wx.ICON_QUESTION)
            val = dlg.ShowModal()
            if val == wx.ID_YES:
                self.OnSaveFile(event)
                self.DoOpenFile()
            elif val == wx.ID_CANCEL:
                dlg.Destroy()
            else:
                self.DoOpenFile()
        else:
            self.DoOpenFile()

    def DoOpenFile(self):
        wcd = 'All files (*)|*|Editor files (*.ef)|*.ef|'
        dir = os.getcwd()
        open_dlg = wx.FileDialog(self, message='Choose a file',
defaultDir=dir, defaultFile='',
                        wildcard=wcd, style=wx.OPEN|wx.CHANGE_DIR)
        if open_dlg.ShowModal() == wx.ID_OK:
            path = open_dlg.GetPath()

            try:
                file = open(path, 'r')
                text = file.read()
                file.close()
                if self.text.GetLastPosition():
                    self.text.Clear()
                self.text.WriteText(text)
                self.last_name_saved = path
                self.statusbar.SetStatusText('', 1)
                self.modify = False
```

```
            except IOError, error:
                dlg = wx.MessageDialog(self, 'Error opening file\n' +
str(error))
                dlg.ShowModal()

            except UnicodeDecodeError, error:
                dlg = wx.MessageDialog(self, 'Error opening file\n' +
str(error))
                dlg.ShowModal()

        open_dlg.Destroy()

    def OnSaveFile(self, event):
        if self.last_name_saved:

            try:
                file = open(self.last_name_saved, 'w')
                text = self.text.GetValue()
                file.write(text)
                file.close()
                self.statusbar.SetStatusText(os.path.basename(self.last_name
_saved) + ' saved', 0)
                self.modify = False
                self.statusbar.SetStatusText('', 1)

            except IOError, error:
                dlg = wx.MessageDialog(self, 'Error saving file\n' +
str(error))
                dlg.ShowModal()
        else:
            self.OnSaveAsFile(event)

    def OnSaveAsFile(self, event):
        wcd='All files(*)|*|Editor files (*.ef)|*.ef|'
        dir = os.getcwd()
        save_dlg = wx.FileDialog(self, message='Save file as...',
defaultDir=dir, defaultFile='',
                        wildcard=wcd, style=wx.SAVE | wx.OVERWRITE_PROMPT)
        if save_dlg.ShowModal() == wx.ID_OK:
            path = save_dlg.GetPath()

            try:
                file = open(path, 'w')
                text = self.text.GetValue()
                file.write(text)
                file.close()
                self.last_name_saved = os.path.basename(path)
                self.statusbar.SetStatusText(self.last_name_saved + '
saved', 0)
                self.modify = False
                self.statusbar.SetStatusText('', 1)

            except IOError, error:
                dlg = wx.MessageDialog(self, 'Error saving file\n' +
str(error))
                dlg.ShowModal()
```

```
        save_dlg.Destroy()

    def OnCut(self, event):
        self.text.Cut()

    def OnCopy(self, event):
        self.text.Copy()

    def OnPaste(self, event):
        self.text.Paste()

    def QuitApplication(self, event):
        if self.modify:
            dlg = wx.MessageDialog(self, 'Save before Exit?', '', wx.YES_NO
| wx.YES_DEFAULT |
                        wx.CANCEL | wx.ICON_QUESTION)
            val = dlg.ShowModal()
            if val == wx.ID_YES:
                self.OnSaveFile(event)
                if not self.modify:
                    wx.Exit()
            elif val == wx.ID_CANCEL:
                dlg.Destroy()
            else:
                self.Destroy()
        else:
            self.Destroy()

    def OnDelete(self, event):
        frm, to = self.text.GetSelection()
        self.text.Remove(frm, to)

    def OnSelectAll(self, event):
        self.text.SelectAll()

    def OnTextChanged(self, event):
        self.modify = True
        self.statusbar.SetStatusText(' modified', 1)
        event.Skip()

    def OnKeyDown(self, event):
        keycode = event.GetKeyCode()
        if keycode == wx.WXK_INSERT:
            if not self.replace:
                self.statusbar.SetStatusText('INS', 2)
                self.replace = True
            else:
                self.statusbar.SetStatusText('', 2)
                self.replace = False
        event.Skip()

    def ToggleStatusBar(self, event):
        if self.statusbar.IsShown():
            self.statusbar.Hide()
        else:
            self.statusbar.Show()
```

```
    def StatusBar(self):
        self.statusbar = self.CreateStatusBar()
        self.statusbar.SetFieldsCount(3)
        self.statusbar.SetStatusWidths([-5, -2, -1])

    def OnAbout(self, event):
        dlg = wx.MessageDialog(self, '\tEditor\t\n Another Tutorial\njan
bodnar 2005-2006',
                                'About Editor', wx.OK | wx.ICON_INFORMATION)
        dlg.ShowModal()
        dlg.Destroy()

app = wx.App()
Editor(None, -1, 'Editor')
app.MainLoop()
```

Kika

Kika es un gript que conecta a un sitio ftp. Si el registro es exitoso, Kika muestra un icono de conectado sobre la barra de estado. De otra manera, un icono de desconectado es mostrado. Usamos el módulo ftplib desde la biblioteca estándar de Python. Si usted no tienen una cuenta ftp, usted puede intentar registrarse a algunos sitios ftp anónimos.



Figura: Kika

```
#!/usr/bin/python
# kika.py

from ftplib import FTP, all_errors
import wx

class MyStatusBar(wx.StatusBar):
    def __init__(self, parent):
        wx.StatusBar.__init__(self, parent)

        self.SetFieldsCount(2)
        self.SetStatusText('Welcome to Kika', 0)
        self.SetStatusWidths([-5, -2])
```

```
        self.icon = wx.StaticBitmap(self, -1,
wx.Bitmap('icons/disconnected.png'))
        self.Bind(wx.EVT_SIZE, self.OnSize)
        self.PlaceIcon()

    def PlaceIcon(self):
        rect = self.GetFieldRect(1)
        self.icon.SetPosition((rect.x+3, rect.y+3))

    def OnSize(self, event):
        self.PlaceIcon()


class Kika(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(250, 270))

        wx.StaticText(self, -1, 'Ftp site', (10, 20))
        wx.StaticText(self, -1, 'Login', (10, 60))
        wx.StaticText(self, -1, 'Password', (10, 100))

        self.ftpsite = wx.TextCtrl(self, -1, '',  (110, 15), (120, -1))
        self.login = wx.TextCtrl(self, -1, '',  (110, 55), (120, -1))
        self.password = wx.TextCtrl(self, -1, '',  (110, 95), (120, -1),
style=wx.TE_PASSWORD)

        self.ftp = None

        con = wx.button(self, 1, 'Connect', (10, 160))
        discon = wx.button(self, 2, 'DisConnect', (120, 160))

        self.Bind(wx.EVT_BUTTON, self.OnConnect, id=1)
        self.Bind(wx.EVT_BUTTON, self.OnDisConnect, id=2)

        self.statusbar = MyStatusBar(self)
        self.SetStatusBar(self.statusbar)
        self.Centre()
        self.Show()

    def OnConnect(self, event):
        if not self.ftp:
            ftpsite = self.ftpsite.GetValue()
            login = self.login.GetValue()
            password = self.password.GetValue()

            try:
                self.ftp = FTP(ftpsite)
                var = self.ftp.login(login, password)
                self.statusbar.SetStatusText('user connected')
                self.statusbar.icon.SetBitmap(wx.Bitmap('icons/connected.png
'))

            except AttributeError:
                self.statusbar.SetForegroundColour(wx.RED)
                self.statusbar.SetStatusText('Incorrect params')
                self.ftp = None
```

```
            except all_errors, err:
                self.statusbar.SetStatusText(str(err))
                self.ftp = None

    def OnDisConnect(self, event):
        if self.ftp:
            self.ftp.quit()
            self.ftp = None
            self.statusbar.SetStatusText('user disconnected')
            self.statusbar.icon.SetBitmap(wx.Bitmap('icons/disconnected.png'
))


app = wx.App()
Kika(None, -1, 'Kika')
app.MainLoop()
```

Tenga en cuenta que cada vez que la ventana es redimensionada, debemos posicionar nuestros iconos a un nuevo lugar.

```
def PlaceIcon(self):
    rect = self.GetFieldRect(1)
    self.icon.SetPosition((rect.x+3, rect.y+3))
```

Puzzle

En este gript, introducimos un juego de puzzle. Tenemos una imagen del personaje Sid desde la pelicula la Edad del Hielo. Éste esta cortado dentro de 9 piezas y mezclado. El objetivo es formar la imagen.



Figura: Puzzle

```
#!/usr/bin/python
# puzzle.py

import wx
import random

class Puzzle(wx.Dialog):
    def __init__(self, parent, id, title):
        wx.Dialog.__init__(self, parent, id, title)

        images = ['images/one.jpg', 'images/two.jpg', 'images/tres.jpg',
'images/cuatro.jpg',
                'images/five.jpg', 'images/six.jpg', 'images/seven.jpg',
'images/eight.jpg']

        self.pos = [ [0, 1, 2], [3, 4, 5], [6, 7, 8] ]

        self.sizer = wx.GridSizer(3, 3, 0, 0)

        numbers = [0, 1, 2, 3, 4, 5, 6, 7]
        random.shuffle(numbers)

        for i in numbers:
                button = wx.BitmapButton(self, i, wx.Bitmap(images[i]))
                button.Bind(wx.EVT_BUTTON, self.OnPressButton,
id=botón.GetId())
                self.sizer.Add(botón)

        self.panel = wx.button(self, -1, size=(112, 82))
        self.sizer.Add(self.panel)

        self.SetSizerAndFit(self.sizer)
        self.Centre()
        self.ShowModal()
        self.Destroy()

    def OnPressButton(self, event):

        button = event.GetEventObject()
```

```
        sizeX = self.panel.GetSize().x
        sizeY = self.panel.GetSize().y

        buttonX = botón.GetPosition().x
        buttonY = botón.GetPosition().y
        panelX = self.panel.GetPosition().x
        panelY = self.panel.GetPosition().y
        buttonPosX = buttonX / sizeX
        buttonPosY = buttonY / sizeY

        buttonIndex = self.pos[buttonPosY][buttonPosX]
        if (buttonX == panelX) and (panelY - buttonY) == sizeY:
            self.sizer.Remove(self.panel)
            self.sizer.Remove(botón)
            self.sizer.Insert(buttonIndex, self.panel)
            self.sizer.Insert(buttonIndex+3, botón)
            self.sizer.Layout()

        if (buttonX == panelX) and (panelY - buttonY) == -sizeY:
            self.sizer.Remove(self.panel)
            self.sizer.Remove(botón)
            self.sizer.Insert(buttonIndex-3, botón)
            self.sizer.Insert(buttonIndex, self.panel)
            self.sizer.Layout()

        if (buttonY == panelY) and (panelX - buttonX) == sizeX:
            self.sizer.Remove(self.panel)
            self.sizer.Remove(botón)
            self.sizer.Insert(buttonIndex, self.panel)
            self.sizer.Insert(buttonIndex+1, botón)
            self.sizer.Layout()

        if (buttonY == panelY) and (panelX - buttonX) == -sizeX:
            self.sizer.Remove(self.panel)
            self.sizer.Remove(botón)
            self.sizer.Insert(buttonIndex-1, botón)
            self.sizer.Insert(buttonIndex, self.panel)
            self.sizer.Layout()

app = wx.App()
Puzzle(None, -1, 'Puzzle')
app.MainLoop()
```

```
images = ['images/one.jpg', 'images/two.jpg', 'images/tres.jpg',
'images/cuatro.jpg',
        'images/five.jpg', 'images/six.jpg', 'images/seven.jpg',
'images/eight.jpg']
```

La imagen fue cortada en 9 partes de tamaño 100x70. Lo hice con el programa Gimp. Cada parte de la imagen es colocada sobre un componente botón. Excepto una.

```
self.sizer = wx.GridSizer(3, 3, 0, 0)
```

Para este gript, *wx.GridSizer* se ajusta idealmente.

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7]
random.shuffle(numbers)
```

Tenemos ocho números. Estos números son mezclar de modo que tenemos una orden de números aleatorios. Cada vez que comenzamos el gript, tendremos un orden diferente para los mapas de bits.

```
self.panel = wx.button(self, -1, size=(112, 82))
self.sizer.Add(self.panel)
```

Este botón no tiene bitmap. Éste es el 'viajero' botón. Éste siempre intercambia su posición con el botón seleccionado.

En este capítulo, hemos presentado varios gripts.

El juego Tetris en wxPython

El juego Tetris es uno de los juegos de computadora mas populares jamas creado. El juego original fue diseñado y programmed por el programador Ruso **Alexey Pajitnov** en 1985. Desde entonces, tetris es disponible sobre casi cualquier plataforma computadora en gran cantidad de variaciones. Aún mi teléfono movil tiene una version modificada del juego Tetris.

Tetris es llamada juego de rompecabezas de bloques cayendo. En este juego, tenemos siete diferentes figuras llamada **tetrominoes**. S-shape, Z-shape, T-shape, L-shape, Line-shape, MirroredL-shape y un box-shape. Cada de estos figuras es formada con cuatro cuadrados. Las figuras van cayendo hacia abajo en el tablero. El objeto del juego Tetris es mover y rotar las figuras, de modo que aquellos llenen tanto como sea posible. Si logramos formar una fila, el fila es destruida y anotamos. Reproducimos el juego Tetris hasta que arribar.

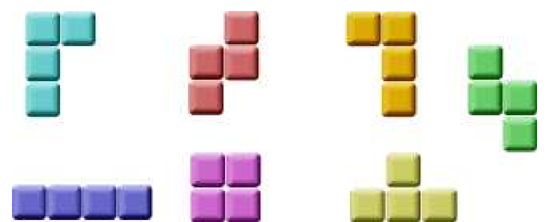


Figura: Tetrominoes

wxPython es un kit de herramientas diseñado para crear aplicaciones. Estas son otras bibliotecas los cuales tienen como objetivo crear juego de computadoras. Sin embargo, wxPython y otras kits de herramientas de aplicaciones pueden ser usado para crear juegos.

El desarrollo

No tenemos que hacer imágenes para nuestras juego Tetris, dibujamos el tetrominoes usando las API de dibujo disponibles en el kit de herramientas de programación de wxPython. Detrás de cada juego de computadora, hay un modelo matemáticas. Pero éste es en Tetris.

Algunos ideas detras el juego.

- usamos **wx.Timer** para crear el ciclo del juego
- Los tetrominoes son dibujado

- Las figuras se mueven cuadro a cuadro (no pixel por pixel)
- Matematicamente un tablero es una simple lista de números

El siguientes ejemplo es una version modificada del juego Tetris, disponibles con los archivos de instalación de PyQt4.

```
#!/usr/bin/python
# tetris.py

import wx
import random

class Tetris(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(180, 380))

        self.statusbar = self.CreateStatusBar()
        self.statusbar.SetStatusText('0')
        self.board = Board(self)
        self.board.SetFocus()
        self.board.start()

        self.Centre()
        self.Show(True)


class Board(wx.Panel):
    BoardWidth = 10
    BoardHeight = 22
    Speed = 300
    ID_TIMER = 1

    def __init__(self, parent):
        wx.Panel.__init__(self, parent)

        self.timer = wx.Timer(self, Board.ID_TIMER)
        self.isWaitingDespués deLine = False
        self.curPiece = Shape()
        self.nextPiece = Shape()
        self.curX = 0
        self.curY = 0
        self.numLinesRemoved = 0
        self.board = []

        self.isStarted = False
        self.isPaused = False

        self.Bind(wx.EVT_PAINT, self.OnPaint)
        self.Bind(wx.EVT_KEY_DOWN, self.OnKeyDown)
        self.Bind(wx.EVT_TIMER, self.OnTimer, id=Board.ID_TIMER)

        self.clearBoard()

    def shapeAt(self, x, y):
        return self.board[(y * Board.BoardWidth) + x]
```

```
    def setShapeAt(self, x, y, shape):
        self.board[(y * Board.BoardWidth) + x] = shape

    def squareWidth(self):
        return self.GetClientSize().GetWidth() / Board.BoardWidth

    def squareHeight(self):
        return self.GetClientSize().GetHeight() / Board.BoardHeight

    def start(self):
        if self.isPaused:
            return

        self.isStarted = True
        self.isWaitingDespués deLine = False
        self.numLinesRemoved = 0
        self.clearBoard()

        self.newPiece()
        self.timer.Start(Board.Speed)

    def pausar(self):
        if not self.isStarted:
            return

        self.isPaused = not self.isPaused
        barra de estado = self.GetParent().statusbar

        if self.isPaused:
            self.timer.Stop()
            statusbar.SetStatusText('paused')
        else:
            self.timer.Start(Board.Speed)
            statusbar.SetStatusText(str(self.numLinesRemoved))

        self.Refresh()

    def clearBoard(self):
        for i in range(Board.BoardHeight * Board.BoardWidth):
            self.board.append(Tetrominoes.NoShape)

    def OnPaint(self, event):

        dc = wx.PaintDC(self)

        size = self.GetClientSize()
        boardTop = size.GetHeight() - Board.BoardHeight *
self.squareHeight()

        for i in range(Board.BoardHeight):
            for j en range(Board.BoardWidth):
                shape = self.shapeAt(j, Board.BoardHeight - i - 1)
                if shape != Tetrominoes.NoShape:
                    self.drawSquare(dc,
                        0 + j * self.squareWidth(),
```

```
                        boardTop + i * self.squareHeight(), shape)

        if self.curPiece.shape() != Tetrominoes.NoShape:
            for i in range(4):
                x = self.curX + self.curPiece.x(i)
                y = self.curY - self.curPiece.y(i)
                self.drawSquare(dc, 0 + x * self.squareWidth(),
                    boardTop + (Board.BoardHeight - y - 1) *
self.squareHeight(),
                    self.curPiece.shape())


    def OnKeyDown(self, event):
        if not self.isStarted o self.curPiece.shape() ==
Tetrominoes.NoShape:
            event.Skip()
            return

        keycode = event.GetKeyCode()

        if keycode == ord('P') o keycode == ord('p'):
            self.pause()
            return
        if self.isPaused:
            return
        elif keycode == wx.WXK_LEFT:
            self.tryMove(self.curPiece, self.curX - 1, self.curY)
        elif keycode == wx.WXK_RIGHT:
            self.tryMove(self.curPiece, self.curX + 1, self.curY)
        elif keycode == wx.WXK_DOWN:
            self.tryMove(self.curPiece.rotatedRight(), self.curX, self.curY)
        elif keycode == wx.WXK_UP:
            self.tryMove(self.curPiece.rotatedLeft(), self.curX, self.curY)
        elif keycode == wx.WXK_SPACE:
            self.dropDown()
        elif keycode == ord('D') o keycode == ord('d'):
            self.oneLineDown()
        else:
            event.Skip()


    def OnTimer(self, event):
        if event.GetId() == Board.ID_TIMER:
            if self.isWaitingDespués deLine:
                self.isWaitingDespués deLine = False
                self.newPiece()
            else:
                self.oneLineDown()
        else:
            event.Skip()


    def dropDown(self):
        newY = self.curY
        while newY > 0:
            if not self.tryMove(self.curPiece, self.curX, newY - 1):
```

```
                break
            newY -= 1

        self.pieceDropped()

    def oneLineDown(self):
        if not self.tryMove(self.curPiece, self.curX, self.curY - 1):
            self.pieceDropped()


    def pieceDropped(self):
        for i in range(4):
            x = self.curX + self.curPiece.x(i)
            y = self.curY - self.curPiece.y(i)
            self.setShapeAt(x, y, self.curPiece.shape())

        self.removeFullLines()

        if not self.isWaitingDespués deLine:
            self.newPiece()


    def removeFullLines(self):
        numFullLines = 0

        barra de estado = self.GetParent().statusbar

        rowsToRemove = []

        for i in range(Board.BoardHeight):
            n = 0
            for j en range(Board.BoardWidth):
                if not self.shapeAt(j, i) == Tetrominoes.NoShape:
                    n = n + 1

            if n == 10:
                rowsToRemove.append(i)

        rowsToRemove.reverse()

        for m en rowsToRemove:
            for k en range(m, Board.BoardHeight):
                for l en range(Board.BoardWidth):
                        self.setShapeAt(l, k, self.shapeAt(l, k + 1))

            numFullLines = numFullLines + len(rowsToRemove)

            if numFullLines > 0:
                self.numLinesRemoved = self.numLinesRemoved + numFullLines
                statusbar.SetStatusText(str(self.numLinesRemoved))
                self.isWaitingDespués deLine = True
                self.curPiece.setShape(Tetrominoes.NoShape)
                self.Refresh()


    def newPiece(self):
```

```
        self.curPiece = self.nextPiece
        barra de estado = self.GetParent().statusbar
        self.nextPiece.setRandomShape()
        self.curX = Board.BoardWidth / 2 + 1
        self.curY = Board.BoardHeight - 1 + self.curPiece.minY()

        if not self.tryMove(self.curPiece, self.curX, self.curY):
            self.curPiece.setShape(Tetrominoes.NoShape)
            self.timer.Stop()
            self.isStarted = False
            statusbar.SetStatusText('Game over')

    def tryMove(self, newPiece, newX, newY):
        for i in range(4):
            x = newX + newPiece.x(i)
            y = newY - newPiece.y(i)
            if x < 0 o x >= Board.BoardWidth o y < 0 o y >=
Board.BoardHeight:
                return False
            if self.shapeAt(x, y) != Tetrominoes.NoShape:
                return False

        self.curPiece = newPiece
        self.curX = newX
        self.curY = newY
        self.Refresh()
        return True


    def drawSquare(self, dc, x, y, shape):
        colors = ['#000000', '#CC6666', '#66CC66', '#6666CC',
                  '#CCCC66', '#CC66CC', '#66CCCC', '#DAAA00']

        light = ['#000000', '#F89FAB', '#79FC79', '#7979FC',
                 '#FCFC79', '#FC79FC', '#79FCFC', '#FCC600']

        dark = ['#000000', '#803C3B', '#3B803B', '#3B3B80',
                 '#80803B', '#803B80', '#3B8080', '#806200']

        pen = wx.Pen(light[shape])
        pen.SetCap(wx.CAP_PROJECTING)
        dc.SetPen(pen)

        dc.DrawLine(x, y + self.squareHeight() - 1, x, y)
        dc.DrawLine(x, y, x + self.squareWidth() - 1, y)

        darkpen = wx.Pen(dark[shape])
        darkpen.SetCap(wx.CAP_PROJECTING)
        dc.SetPen(darkpen)

        dc.DrawLine(x + 1, y + self.squareHeight() - 1,
            x + self.squareWidth() - 1, y + self.squareHeight() - 1)
        dc.DrawLine(x + self.squareWidth() - 1,
        y + self.squareHeight() - 1, x + self.squareWidth() - 1, y + 1)

        dc.SetPen(wx.TRANSPARENT_PEN)
```

```
        dc.SetBrush(wx.Brush(colors[shape]))
        dc.DrawRectangle(x + 1, y + 1, self.squareWidth() - 2,
        self.squareHeight() - 2)


class Tetrominoes(object):
    NoShape = 0
    ZShape = 1
    SShape = 2
    LineShape = 3
    TShape = 4
    SquareShape = 5
    LShape = 6
    MirroredLShape = 7


class Shape(object):
    coordsTable = (
        ((0, 0),     (0, 0),     (0, 0),     (0, 0)),
        ((0, -1),    (0, 0),     (-1, 0),    (-1, 1)),
        ((0, -1),    (0, 0),     (1, 0),     (1, 1)),
        ((0, -1),    (0, 0),     (0, 1),     (0, 2)),
        ((-1, 0),    (0, 0),     (1, 0),     (0, 1)),
        ((0, 0),     (1, 0),     (0, 1),     (1, 1)),
        ((-1, -1),   (0, -1),    (0, 0),     (0, 1)),
        ((1, -1),    (0, -1),    (0, 0),     (0, 1))
    )

    def __init__(self):
        self.coords = [[0,0] for i in range(4)]
        self.pieceShape = Tetrominoes.NoShape

        self.setShape(Tetrominoes.NoShape)

    def shape(self):
        return self.pieceShape

    def setShape(self, shape):
        table = Shape.coordsTable[shape]
        for i in range(4):
            for j en range(2):
                self.coords[i][j] = table[i][j]

        self.pieceShape = shape

    def setRandomShape(self):
        self.setShape(random.randint(1, 7))

    def x(self, index):
        return self.coords[index][0]

    def y(self, index):
        return self.coords[index][1]

    def setX(self, index, x):
        self.coords[index][0] = x
```

```
    def setY(self, index, y):
        self.coords[index][1] = y

    def minX(self):
        m = self.coords[0][0]
        for i in range(4):
            m = min(m, self.coords[i][0])

        return m

    def maxX(self):
        m = self.coords[0][0]
        for i in range(4):
            m = max(m, self.coords[i][0])

        return m

    def minY(self):
        m = self.coords[0][1]
        for i in range(4):
            m = min(m, self.coords[i][1])

        return m

    def maxY(self):
        m = self.coords[0][1]
        for i in range(4):
            m = max(m, self.coords[i][1])

        return m

    def rotatedLeft(self):
        if self.pieceShape == Tetrominoes.SquareShape:
            return self

        result = Shape()
        result.pieceShape = self.pieceShape
        for i in range(4):
            result.setX(i, self.y(i))
            result.setY(i, -self.x(i))

        return result

    def rotatedRight(self):
        if self.pieceShape == Tetrominoes.SquareShape:
            return self

        result = Shape()
        result.pieceShape = self.pieceShape
        for i in range(4):
            result.setX(i, -self.y(i))
            result.setY(i, self.x(i))

        return result
```

```
app = wx.App()
Tetris(None, -1, 'Tetris')
app.MainLoop()
```

He simplificado el juego un poquito, de modo que éste sea fácil para comprender. El juego arranca inmediatamente, Después de éste es lanzado. Podemos pausar el juego presionando la tecla p. La barra espaciadora podrá caer la pieza del tetris inmediatamente hacia el fondo. El tecla d podrá caer la pieza uno línea abajo. (Esto puede ser usado para acelerar la caida un poquito.) el juego va velocidad constante, no se implementó acceleración. El marcador es el número de líneas, que tenemos removidas.

```
...
self.curX = 0
self.curY = 0
self.numLinesRemoved = 0
self.board = []
...
```

Antes que comencemos un cliclo del juego, inicializamos algunas variables importantes. La variable **self.board** es una lista de números de 0 a 7. Ésta representa la posición de varios figuras y remanente de la formas sobre el tablero.

```
for i in range(Board.BoardHeight):
    for j en range(Board.BoardWidth):
        shape = self.shapeAt(j, Board.BoardHeight - i - 1)
        if shape != Tetrominoes.NoShape:
            self.drawSquare(dc,
                0 + j * self.squareWidth(),
                boardTop + i * self.squareHeight(), shape)
```

El pintado del juego es dividido en dos pasos. En el primer paso, dibujamos todas las figuras, o remanente de las formas, que han sido retiradas del fondo del tablero. Todos el cuadrados son recordados en la variable lista del **self.board**. Lo accedemos usando el método shapeAt().

```
if self.curPiece.shape() != Tetrominoes.NoShape:
    for i in range(4):
        x = self.curX + self.curPiece.x(i)
        y = self.curY - self.curPiece.y(i)
        self.drawSquare(dc, 0 + x * self.squareWidth(),
            boardTop + (Board.BoardHeight - y - 1) * self.squareHeight(),
            self.curPiece.shape())
```

El siguiente paso es dibujar la pieza actual, que está cayendo.

```
elif keycode == wx.WXK_LEFT:
    self.tryMove(self.curPiece, self.curX - 1, self.curY)
```

En el método **OnKeyDown()** comprobamos por las teclas pulsadas. Si pulsamos la tecla izquierda flecha, intentamos mover la pieza a la izquierda. Decimos intentar, porque la pieza podría no estar disponible para mover.

```
def tryMove(self, newPiece, newX, newY):
    for i in range(4):
        x = newX + newPiece.x(i)
        y = newY - newPiece.y(i)
```

```
        if x < 0 o x >= Board.BoardWidth o y < 0 o y >= Board.BoardHeight:
            return False
        if self.shapeAt(x, y) != Tetrominoes.NoShape:
            return False

    self.curPiece = newPiece
    self.curX = newX
    self.curY = newY
    self.Refresh()
    return True
```

En el método **tryMove()** intentamos mover nuestras figuras. Si la figura está en el borde del tablero o es adjacente a algunas otras pieza, retornamos false. De otra manera colocamos la pieza que está ahora cayendo en una nueva posición y retornamos true.

```
def OnTimer(self, event):
    if event.GetId() == Board.ID_TIMER:
        if self.isWaitingDespués deLine:
            self.isWaitingDespués deLine = False
            self.newPiece()
        else:
            self.oneLineDown()
    else:
        event.Skip()
```

En el método **OnTimer()** o creaamos una nueva pieza, después de que una llego al fondo, o movemos la pieza que cae una línea abajo.

```
def removeFullLines(self):
    numFullLines = 0

    rowsToRemove = []

    for i in range(Board.BoardHeight):
        n = 0
        for j en range(Board.BoardWidth):
            if not self.shapeAt(j, i) == Tetrominoes.NoShape:
                n = n + 1

        if n == 10:
            rowsToRemove.append(i)

     rowsToRemove.reverse()

     for m en rowsToRemove:
         for k en range(m, Board.BoardHeight):
             for l en range(Board.BoardWidth):
                 self.setShapeAt(l, k, self.shapeAt(l, k + 1))
...
```

Si la pieza toca el fondo, llamamos el método **removeFullLines()**. Primero nos encontramos con todos las líneas llenas. Y las removemos. Lo hacemos moviendo todas los líneas abajo la línea llena a ser removida. Note que preservamos el orden de las líneas a ser removidas. De otra manera, podría no trabajar correctamente. En nuestro caso usamos una **gravedad ingenua**. De esta manera, que las piezas pueden ser flotar hacia abajo varios pasos.

```
def newPiece(self):
    self.curPiece = self.nextPiece
    barra de estado = self.GetParent().statusbar
    self.nextPiece.setRandomShape()
    self.curX = Board.BoardWidth / 2 + 1
    self.curY = Board.BoardHeight - 1 + self.curPiece.minY()

    if not self.tryMove(self.curPiece, self.curX, self.curY):
        self.curPiece.setShape(Tetrominoes.NoShape)
        self.timer.Stop()
        self.isStarted = False
        statusbar.SetStatusText('Game over')
```

El método **newPiece()** crea al azar una nueva pieza del Tetris. Si la pieza no entra en su posición inicial, el juego termina.

La clase **Shape** guarda información acerca de la pieza del Tetris.

```
self.coords = [[0,0] for i in range(4)]
```

Al momento de la creación, creamos una lista vacia de coordenadas. La lista podrá guardar las coordenadas de las piezas del Tetris. Por ejemplo, estos tupla (0, -1), (0, 0), (1, 0), (1, 1) representan una S-shape rotada. El siguiente diagrama ilustra la figura.

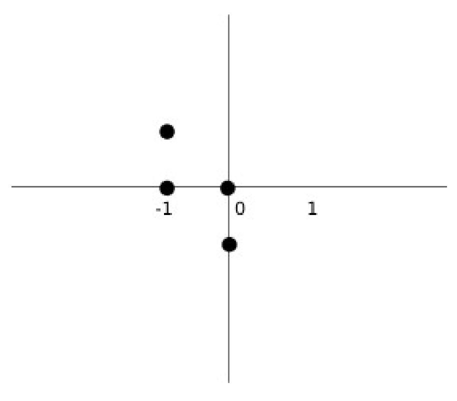


Figura: Coordinates

Cuando dibujamos la pieza que esta cayendo ahora, la dibujamos en la posición **self.curX**, **self.curY** . Entonces nosotros miramos a la tabla de coordenadas y dibujamos todos los el cuatro cuadrado.

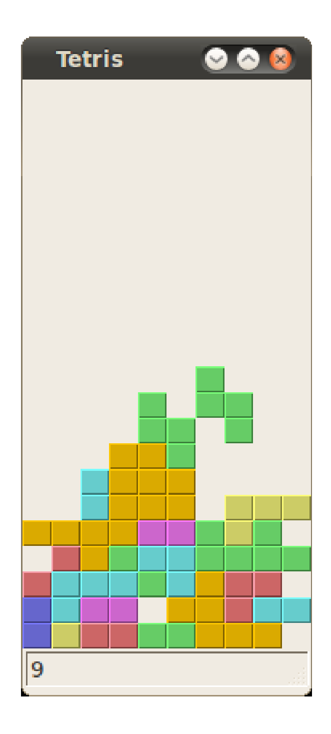


Figura: Tetris

Este fue un juego Tetris en wxPython.

Este documento fue traducido por Carlos Miguel FARÍAS. Santa Rosa – La Pampa - Argentina

No está traducido con programa traductor, se uso el traductor de Google para algunas frases y palabras en general.

El documento original está indicado al pie de cada página.

Favor de informar si hay algún error así se corrige.